Technische Universität Darmstadt, Institute of Microelectronic Systems

Algorithm, Application Mapping, Design and Realization of the Time-Frequency Representation with Flexible Kernels based on their Lifting Scheme

Andre Guntoro

Ph.D. Thesis, June 2009

Institute of Microelectronic Systems
Technische Universität Darmstadt
Karlstr. 15, D-64283 Darmstadt, Germany

Algorithm, Application Mapping, Design and Realization of the Time-Frequency Representation with Flexible Kernels based on their Lifting Scheme

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

M.Sc.
Andre T. Guntoro
geboren am 13. Juli 1979
in Indramayu, Indonesien

Referent:	Prof. Dr. Dr. h.c. mult. Manfred Glesner *Technische Universität Darmstadt*
Korreferent:	Prof. Dr.-Ing. Dr. h.c. Norbert Fliege *Universität Mannheim*
Tag der Einreichung:	30.06.2009
Tag der mündlichen Prüfung:	05.11.2009

D17

Darmstadt 2009

buat papa. . .

"There must be a beginning of any great matter,
but the continuing unto the end until it be
thoroughly finished yields the true glory."

Sir Francis Drake

Acknowledgments

My very first thank you goes to Jesus Christ for his unconditional loving. Without your passion and endless blessing, I won't be able to stand strong and accomplish many great achievements.

I'd like to thank Prof. Manfred Glesner for his guidance and for giving me the opportunity to be involved in various projects. Also I'd like to thank Prof. Norbert Fliege who kindly accepted to act as a reviewer for this thesis. Further I'd like to thank Prof. Abdelhak Zoubir, Prof. Jürgen Adamy and Prof. Gerd Balzer for acting as members of the examination committee.

I'd like to express my sincerely thanks to all colleagues at the institute for giving me such a warm working atmosphere. For the support and kindness that I tasted during my work, I'd like to express my gratitude to Oana Mutihac, Massoud Momeni, Hans-Peter Keil, Tudor Murgan, Andreas Schmidt, Petru Bacinschi, Ping Zhao, Oliver Soffke and Heiko Hinkelmann. Also I'd like to thank the older and former colleagues Lukusa Kabulepa and Matthias Rychetsky for their advices.

Living in a foreign country is full of challanges. That's why I'd like to specifically thank Alvin Kurnadi, Ellis Bong, Christina Wibisono, Nicole Irwanto, Conny Man, Susi Schmidt, Hendrina Pattiradjawane and Henny Schwöbel for being there for me. Especially, I'd like to thank Serena Suprawoto for being a good friend since the old days and for her excellent English expertise. For the warmth welcome, spiritual supports, kindness, friendship and good foods of course, I'd like to express my gratefulness to Christian community PERKI Darmstadt (now JKI). Also I'd like to thank Jochen Springmann for giving me advices and for sharing your worries. My special thank you goes to Thomas Rink for listening to my problems and burdens, supporting me through the years and bringing the new felicity everyday.

I'd like to dedicate this thesis to my late father. Thank you for not giving hope on me and believing in me. I'd like to greatly thank my mother, brothers and sisters for their love, unconditional supports, care, trust, inspiration and encouragement. Thank you for giving me the best place to grow up. Without you and your education I don't know what I would be.

Darmstadt, November 2009

Abstract

Wavelets have become a hot topic in both industry and research fields in the recent years. In the transform block of JPEG2000, two different wavelet filters can be applied depending on the compression methods: (5,3) for lossless and (9,7) for lossy compression. Besides the block transform of JPEG2000, wavelet transforms are applied in many other applications, such as feature detection, voice synthesis, statistic, etc. The major challenge in the wavelet transforms is that there exist different classes of wavelet filters for different kinds of applications.

In this thesis, we propose generalized lifting-based wavelet processors that can perform various DWT decompositions and reconstructions, as well as DWP decompositions and reconstructions with different types of wavelet filters. The processors are based on cross-chained processing elements which perform prediction and update atom functions of the lifting-based transforms. Two different arithmetics are designed in order to adapt with diversities in applications' demand: fixed-point and floating-point wavelet processors. On each type of arithmetic, two architectures are proposed: resource-aware architecture which exploits time-sharing property of the arithmetic units and has processing speed of $f/2$, and high-performance architecture which uses dedicated arithmetic units and has processing speed of f. The generalization of the proposed wavelet processors extends in many ways. The proposed processors can compute N-dimensional transforms, as well as multilevel transforms for 1D signal. On some applications that require energy conservation during the transforms, we also consider the normalization step which takes place at the end of the decomposition or at the beginning of the reconstruction. Our proposed wavelet processors can also be configured to have arbitrary data width, including the fraction size of the floating-point architectures. Because different applications require different number of samples for the transforms, we propose a flexible memory size that can be implemented in the design. To cope with different wavelet filters, we feature a multi-context configuration to select among various transforms. This context switch is further used as a configuration tool to compute wavelet filters with longer lifting steps.

Our wavelet processors are modelled and synthesized with a parameterizable VHDL code written at the RTL level. The performance of our processors varies depending on the data width selections, the architecture types, and the wavelet filters. For 32-bit resource-aware floating-point architecture, the proposed processor can compute lossless JPEG2000 transform of 512×512 image with 211 fps.

Kurzfassung

In den letzten Jahren wurden Wavelets sowohl in Forschung als auch in der Industrie sehr umfassend untersucht. Im Transformationsblock des JPEG2000 Algorithmus können je nach Kompressionsmethode zwei unterschiedliche Wavelet Filter eingesetzt werden: (5,3) für eine verlustfreie und (9,7) für eine verlustbehaftete Kompression. Außer in der Blocktransformation von JPEG2000 finden Wavelet-Filter ihren Einsatz in vielen weiteren Anwendungen, wie z.B. Mustererkennung, Sprachsynthese, Statistik usw. Die größte Herausforderung besteht darin, dass für unterschiedliche Anwendungen unterschiedliche Klassen von Wavelet-Filter bestehen.

In dieser Dissertation werden verallgemeinerte Lifting-basierte Wavelet-Prozessoren vorgeschlagen, die sowohl verschiedene DWT (Discrete Wavelet Transform) Dekompositionen und Rekonstruktionen als auch DWP (Discrete Wavelet Packet) Dekompositionen und Rekonstruktionen mit verschiedenen Typen von Wavelet-Filter durchführen können. Die Prozessoren basieren auf verketteten Prozesselementen (PE) zur Berechnung und Aktualisierung von Atom-Funktionen in den Lifting-basierten Transformationen. Zwei unterschiedliche Arten der Arithmetik wurden berücksichtigt, um die große Fülle von Anwendungen zu berücksichtigen: Fixpunkt- und Gleitkomma-Wavelet-Prozessoren. Für beide Arten der Arithmetik wurden zwei Prozessoren entworfen: eine Resourcen-sparsame Architektur, die das Zeitteilverfahren der arithmetischen Einheiten ausnutzt und bei einer Verarbeitungs-Geschwindigkeit von $f/2$ betrieben wird und eine hochperformante Architektur, welche dedizierte arithmetische Einheiten einsetzt und bei einer Verarbeitungs-Geschwindigkeit von f betrieben wird. Die Verallgemeinerung der vorgeschlagenen Wavelet-Prozessoren schlägt sich auf viele Arten nieder. Die Prozessoren können sowohl N-dimensionale Transformationen als auch Multi-Level Transformationen eines 1D-Signals berechnen. Für manche Anwendungen, die eine Energieerhaltung während der Transformation diktieren, betrachten wir auch die Normalisierung am Ende der Dekomposition bzw. am Anfang der Rekonstruktion. Die vorgeschlagenen Prozessoren können für eine beliebige Datenbreite konfiguriert werden, auch die Anzahl der Mantissenziffern in der Gleitkommadarstellung kann beliebig eingestellt werden. Da unterschiedliche Anwendungen eine unterschiedliche Anzahl von Samples für die Transformation benötigen, wird eine flexible Speichergröße vorgeschlagen, die im Entwurf implementiert werden kann. Um verschiedene Wavelet-Filter zur Berechnung einer Vielzahl von Transformationen realisieren zu können, wird eine Multi-Kontext Kon-

figuration vorgestellt. Dieser Kontext-Switch wird auch als ein Konfigurations-Tool zur Berechnung von Wavelet-Filter mit längeren Lifting Schritten eingetzt.

Die vorgestellten Wavelet-Prozessoren wurden in parametrisierbarem VHDL-Code auf RTL-Ebene modelliert und synthetisiert. Die Performanz der Prozessoren unterscheiden sich je nach Datenbreite, Architekturtyp und Wavelet-Filter. Für die 32-bit Ressourcen-sparsame Gleitkomma-Architektur kann der Prozessor eine verluftfreie JPEG2000 Transformation eines 512×512 Bildes mit 211 fps berechnen.

Table of Contents

1 Introduction and Overview **1**

1.1 Motivation 1

1.2 Research Scope and Objectives 2

1.3 Thesis Outline 2

2 Wavelets and Wavelet Algorithms **5**

2.1 Time-Frequency Representation 5

2.1.1 The Heisenberg Uncertainty Principle 7

2.2 Short-Time Fourier Transform 9

2.2.1 Gabor Transform 10

2.2.2 Properties of Short-Time Fourier Transform 10

2.2.3 Discrete Representation of STFT 11

2.3 Continuous Wavelet Transform 12

2.4 Properties of Wavelets 13

2.4.1 Admissible Condition 14

2.4.2 Regularity 16

2.4.3 Localization Analysis 16

2.4.4 Wavelet Transform of Basic Functions 17

2.5 Time-Frequency Plane 17

2.6 Discrete Wavelet Transform 19

2.7 Multiresolution Signal Analysis 23

2.8 Two-Scale and Decomposition Relations 25

2.8.1 Relation between Two-Scale Sequences 28

2.8.2 Relation between Reconstruction and Decomposition Sequences . . 29

2.9 Filter Banks 30

2.9.1 Decimation and Interpolation 31

2.9.2 Decomposition and Reconstruction Algorithms 32

2.9.3 Two-Channel Perfect Reconstruction Filter Bank 34

2.10 Polyphase Representation for Filter Banks 36

2.11 Lifting Scheme 37

3 State-of-the-Art in Discrete Wavelet Processors 41
3.1 Filter Banks Systolic-based Architectures 43
3.1.1 DWT-SA Architecture 43
3.1.2 Arc_J Architecture 46
3.2 Lifting-Based Architectures 48
3.2.1 Recursive Architecture 49
3.2.2 1D Folded Architecture 51
3.2.3 Row and Column Processors Architectures 53
3.3 Architectures for Computing Discrete Wavelet Packet 55
3.4 Concluding Remarks 57

4 Architecture Design Consideration 59
4.1 Algorithm and Application Mapping 59
4.2 Observation on the Algorithm for Discrete Wavelet Decomposition and Reconstruction 64
4.2.1 Observation on the Lifting Scheme 64
4.2.2 Wavelet Transform and Wavelet Packet 70
4.3 Design Analysis 71
4.4 Fixed-Point based PE 72
4.4.1 Resource-aware Fixed-Point PE 73
4.4.2 High-Performance Fixed-Point PE 76
4.5 The Need of Floating-Point 79
4.5.1 Standard IEEE 754 Format 81
4.5.2 Floating-Point Multiplier 81
4.5.3 Floating-Point Addition Algorithm 84
4.5.4 4-Stage Floating-Point Adder with Three Inputs 85
4.5.5 3-Stage Floating-Point Adder with Three Inputs 89
4.5.6 Resource-aware Floating-Point PE 92
4.5.7 High-Performance Floating-Point PE 94
4.6 Context Switch for PE 96
4.7 Common Components 100
4.7.1 Main FSM 101
4.7.2 Config 104
4.7.3 Memory 105
4.7.4 Arbiter 108
4.7.5 Source and Sink 108

4.7.6 Latency Counter . . . 110
4.8 Details of the Transform Process . . . 110
4.8.1 1D Discrete Wavelet Decomposition for DWT . . . 111
4.8.2 1D Discrete Wavelet Reconstruction for DWT . . . 114
4.8.3 1D Discrete Wavelet Decomposition for DWP . . . 116
4.8.4 1D Discrete Wavelet Reconstruction for DWP . . . 119
4.8.5 *N*-Dimensional DWT Decomposition and Reconstruction . . . 120
4.8.6 *N*-Dimensional DWP Decomposition and Reconstruction . . . 122
4.9 Concluding Remarks . . . 123

5 Analysis on Performance and Benchmarks 125
5.1 Analysis on Floating-Point Arithmetic Cores . . . 127
5.1.1 2-Stage Floating-Point Multiplier . . . 127
5.1.2 4-Stage Floating-Point Adder . . . 128
5.1.3 3-Stage Floating-Point Adder . . . 130
5.1.4 Discussion on the Floating-Point Cores . . . 131
5.2 Analysis on Different Types of PEs . . . 132
5.3 Analysis on Different Number of PEs . . . 136
5.4 Analysis on Different Memory Organizations . . . 138
5.5 Analysis on Supporting Wavelet Packet . . . 140
5.6 Performances . . . 142
5.6.1 Performances on 1D Signals . . . 142
5.6.2 Performances on 2D Signals . . . 148
5.7 Benchmarks . . . 151
5.8 Comparisons . . . 155
5.9 Concluding Remarks . . . 157

6 Conclusions and Future Works 159
6.1 Contributions of the Work . . . 160
6.2 Directions for Future Work . . . 161

A Fourier Analysis 163
A.1 Fourier Series and Fourier Transform . . . 163
A.2 Discrete-Time Fourier Transform . . . 166
A.3 Discrete Fourier Transform . . . 167

B Orthogonal, Biorthogonal, and Semiorthogonal 169

C Factoring Wavelet Filters to Lifting Steps 173

References **177**

List of Publications **179**

Supervised Theses **181**

List of Abbreviations

CWT	Continuous Wavelet Transform
DFT	Discrete Fourier Transform
DTFT	Discrete-Time Fourier Transform
DWT	Discrete Wavelet Transform
DWP	Discrete Wavelet Packet
EBCOT	Embedded Block Coding with Optimized Truncation
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
JPEG	Joint Photographic Experts Group
LOD	Leading-One Detector
LOP	Leading-One Predictor
MAC	Multiply-and-Accumulate
MRA	Multirate Signal Analysis
PDF	Probability Density Function
PE	Processing Element
OFDM	Orthogonal Frequency Division Multiplexing
SIMD	Single Instruction Multiple Data
SNR	Signal-to-Noise Ratio
STFT	Short-Time Fourier Transform

List of Symbols

ψ	Mother function
ϕ	Scaling function
$\psi_{a,b}$	Wavelet expansions
$\psi_{m,n}$	Discrete wavelet expansions
h	Low-pass synthesis function
g	High-pass synthesis function
$\tilde{h}$	Low-pass analysis function
$\tilde{g}$	High-pass analysis function
$\mathbf{V_j}$	Scaling spaces
$\mathbf{W_j}$	Wavelet spaces
$\mathscr{P}$	Prediction function
$\mathscr{U}$	Update function
$\mathbf{P}$	Polyphase matrix

List of Tables

4.1 The filter coefficients of Daub-12 for the compactly supported wavelet. . . . 63
4.2 Truth table of possible resulting fraction after the unsigned multiplication. . 83
4.3 Description of the parameters stored in configuration block. 105
4.4 Arbitration of the memory bus ownership. 108
4.5 Initialization values. 112

5.1 Estimated area and frequency of the floating-point multiplier. 128
5.2 Estimated area and frequency of the 4-stage floating-point adder. 129
5.3 Estimated area and frequency of the 3-stage floating-point adder. 130
5.4 Fixed-point mapping. 132
5.5 Estimated area and frequency of the normal PEs and the PEs with normalizer that are based on fixed-point arithmetics. 133
5.6 Estimated area and frequency of the normal PEs and the PEs with the normalizer that are based on floating-point arithmetics. 133
5.7 Details of area usage of the fixed-point PEs (all values are in μm^2). 135
5.8 Details of area usage of the floating-point PEs (all values are in μm^2). . . . 135
5.9 Estimated area and frequency of the fixed-point wavelet processors with different numbers of PEs. 137
5.10 Estimated area and frequency of the floating-point wavelet processors with different numbers of PEs. 137
5.11 Estimated area and frequency of the fixed-point wavelet processors with different memory configurations. 138
5.12 Estimated area and frequency of the floating-point wavelet processors with different memory configurations. 139
5.13 Details of area usage of the fixed-point wavelet processors with different memory configurations (all values are in μm^2). 139
5.14 Details of area usage of the floating-point wavelet processors with different memory configurations (all values are in μm^2). 140

5.15 Estimated area and frequency of the fixed-point wavelet processors without and with DWP support. . . . 141
5.16 Estimated area and frequency of the floating-point wavelet processors without and with DWP support. . . . 141
5.17 Decomposition lifting coefficients of (9,7), Daub-4, Daub-6, Symlet-6, and Coiflet-2 wavelet filters. . . . 143
5.18 Benchmarks for 1D signal with (5,3) filter. . . . 152
5.19 Benchmarks for 1D signal with (9,7) filter. . . . 153
5.20 Benchmarks for JPEG2000 with (5,3) filter. . . . 154
5.21 Benchmarks for JPEG2000 with (9,7) filter. . . . 154
5.22 Comparison with other lifting-based architectures. . . . 156

List of Figures

2.1 Two elementary operations on a basis function f and their effects on the time-frequency plane [**?**]. 7
2.2 Time shift and frequency shift operation on STFT and their effect on time-frequency plane [**?**]. 11
2.3 Time-frequency plane of different bases [**?**, **?**, ?]. 18
2.4 Lattice structure of localization of the discrete wavelets in time-frequence plane [?, ?]. 20
2.5 Hierarchical structure of MRA subspaces [?]. 25
2.6 Two-scale relations for Haar wavelet [?]. 27
2.7 Decomposition relations for Haar wavelet [?]. 28
2.8 Diagram of decimator and interpolator [?]. 32
2.9 Decomposition process for a one-level wavelet decomposition tree [?]. . . . 33
2.10 Reconstruction process for a one-level wavelet reconstruction tree [?]. 34
2.11 Two-channel filter bank [?]. 35
2.12 Polyphase realization of a two-channel filter bank [?]. 37
2.13 Basic lifting scheme with prediction and update step [**?**]. 39
2.14 Wavelet decomposition using lifting steps [**?**]. 40

3.1 DWT-SA architecture proposed by Grzeszczak [**?**]. 44
3.2 Filter unit with low-pass and high-pass coefficients selector [?]. 45
3.3 Arc_J architecture for first decomposition level [**?**]. 46
3.4 Arc_J architecture for second decomposition level [?]. 47
3.5 Recursive architecture proposed by Liao [**?**]. 50
3.6 1D folded architecture proposed by Lian [**?**]. 51
3.7 Row and column architecture [**?**]. 54
3.8 Data flow for (5,3) and (9,7) filter mode [?]. 54
3.9 Basic architecture of each processor [?]. 55

3.10 Folded architecture for DWP [**?**]. 56

4.1 Lifting scheme of discrete wavelet decomposition. 66
4.2 Lifting scheme of discrete wavelet reconstruction. 66
4.3 Discrete wavelet decomposition that exercises longer wavelet filter. 67
4.4 Step by step discrete wavelet decomposition. 69
4.5 Lattice structure lifting steps of Daubechies-2 wavelet filter. 69
4.6 Lattice structure of lifting steps of Coiflet-2 wavelet filter. 70
4.7 Wavelet transforms using filter banks. 71
4.8 Wavelet packets using filter banks. 71
4.9 Block diagram of the resource-aware fixed-point PE with one multiplier and one adder. 73
4.10 Multiply-And-Accumulate unit. 75
4.11 Timing diagram of the multiplexer selectors related to the input samples. . 76
4.12 Block diagram of the resource-aware fixed-point PE which is located on the top and on the bottom of wavelet processor. 77
4.13 Block diagram of the high-performance fixed-point PE with two multipliers and two adders. 78
4.14 Block diagram of the high-performance fixed-point PE with two multipliers and two adders and an additional capability to compute the normalization. 80
4.15 Block diagram of the floating-point multiplier. 82
4.16 Two-Input Floating-Point Adder with Leading-One Detector (LOD). 84
4.17 Two-Input Floating-Point Adder with Leading-One Predictor (LOP). 85
4.18 The architecture of 4-stage 3-input floating-point adder. 86
4.19 The architecture of 3-stage 3-input floating-point adder. 90
4.20 Block diagram of the resource-aware floating-point PE with one multiplier and 4-stage adder. 94
4.21 Block diagram of the resource-aware floating-point PE which is located on the top and on the bottom of wavelet processor. 95
4.22 Block diagram of the high-performance floating-point PE with two multipliers and 3-stage adder. 96
4.23 Block diagram of the high-performance floating-point PE which is located on the top and on the bottom of wavelet processor. 97
4.24 Context switch for the PE. 98
4.25 Functional unit that consists of a PE and its context switch. 98

4.26 Cross-chained PEs. 99
4.27 The Proposed Wavelet Processor. 101
4.28 Main FSM that controls the wavelet processor. 102
4.29 The content of the memory using one bank configuration only. 106
4.30 The content of the memory using two banks configuration. 106
4.31 Source FSM. 109
4.32 4-level 1D DWT wavelet decomposition. 111
4.33 3-level 1D DWT wavelet decomposition with multiple lifting steps. 113
4.34 4-level 1D DWT wavelet reconstruction. 114
4.35 3-level 1D DWT wavelet reconstruction with multiple lifting steps. 115
4.36 4-level 1D DWP wavelet decomposition. 116
4.37 Timing diagram of 1D DWP decomposition. 117
4.38 3-level 1D DWP wavelet decomposition with multiple lifting steps. 119
4.39 Timing diagram of 1D DWP decomposition with multiple lifting steps. . . . 119
4.40 4-level 1D DWP wavelet reconstruction. 120
4.41 2-level 2D DWT decomposition. 120
4.42 Process of a N-level 2D DWT decomposition. 121
4.43 First level and second level 2D DWP decomposition. 122

5.1 Estimated frequency and area of floating-point multiplier for various data widths. 128
5.2 Estimated frequency and area of 4-stage floating-point adder for various data widths. 129
5.3 Estimated frequency and area of 3-stage floating-point adder for various data widths. 130
5.4 Estimated frequency of floating-point arithmetics. 131
5.5 Estimated area and estimated frequency of resource-aware fixed-point and floating-point PEs. 134
5.6 Estimated area and estimated frequency of high-performance fixed-point and floating-point PEs. 134
5.7 Multilevel DWT using fixed-point wavelet processors with different data width implementations. 144
5.8 Multilevel DWP using fixed-point wavelet processors with different data width implementations. 145

5.9 Multilevel DWT using floating-point wavelet processors with different data width implementations. 147
5.10 Multilevel DWP using floating-point wavelet processors with different data width implementations. 148
5.11 2D image sources. 149
5.12 SNR values of various 2D images. 150

Chapter 1

Introduction and Overview

Contents

1.1 Motivation 1

1.2 Research Scope and Objectives 2

1.3 Thesis Outline 2

1.1 Motivation

For the last two decades the wavelet theory has been studied extensively [**?**,**?**,**?**,**?**,**?**] to answer the demand for better and more appropriate functions to represent signals than the ones offered by the Fourier analysis. Contrary to the Fourier analysis, which decomposes signals into sine and cosine functions, wavelets study each component of the signal on different resolutions and scales. In analogy, if we observe the signal with a large *window*, we will get a coarse feature of the signal, and if we observe the signal with a small *window*, we will extract the details of the signal.

One of the most attractive features that wavelet transforms provide is their capability to analyze the signals which contain sharp spikes and discontinuities. The better energy compacting support the wavelet transforms offer and also the localizing feature [**?**] of the signal in both time and frequency domains these transforms support have made wavelet outperforms the Fourier transform in signal processing and has made itself into the new standard of JPEG2000 [**?**,**?**].

In the recent years, wavelets have become a hot topic in both industry and research fields due to the adoption of the wavelet transforms by the JPEG committee in their new JPEG2000 standard. In the transform block of JPEG2000, two different wavelet filters can be applied depending on the compression methods. For lossless compression, (5,3) filter is used because it has integer coefficients and requires no scaling, whereas for lossy compression, (9,7) filter is used because it can exploit the locality property of the signal

better compared to (5,3) filter. Along with recent trends and research focuses in applying wavelets in image processing, the application of wavelets is essentially not only limited to this area. The benefits of wavelets have been studied by many scientists from different fields such as mathematics, physics, and electrical engineering. In the field of electrical engineering wavelets have been known with the name multirate signal processing. Due to numerous interchanging fields, wavelets have been used in many applications such as image compression, feature detection, seismic geology, human vision, etc.

Contrary to the Fourier transform, which uses one basis function (and its inverse) to transform between domains, there are different classes of wavelet kernels which can be applied on the signal depending on the application. Because different applications require different treatments, researchers have tried to cope with their own issues and implemented only a subset of wavelets which are suitable for their own needs such as ones that can be found in image compression [**?**, **?**, ?, **?**] and speech processing [**?**, **?**, **?**, **?**]. The power of wavelet tools is then limited due to these approaches.

1.2 Research Scope and Objectives

In this thesis we propose a novel architecture to compute decomposition (forward transform) and reconstruction (inverse transform) of numerous DWTs (Discrete Wavelet Transforms) and also DWPs (Discrete Wavelet Packets) based on their lifting scheme representations. Most lifting-based wavelet processors are dedicated to compute wavelet filters which are used only in JPEG2000 image compression where the wavelet coefficients can be represented as integers and have simple prediction and update functions.

Our new proposed architecture takes into account that each lifting step representation of an arbitrary wavelet filter can be factored to have two prediction or update coefficients, in order to reduce the number of lifting steps. Additionally, the proposed architecture also considers the normalization step which takes place at the end of the DWT/DWP decomposition or at the beginning of the DWT/DWP reconstruction for the applications that require to conserve the energy during the transform. In order to be flexible, the proposed architecture provides a multi-context configuration to choose between various DWT/DWP decomposition and reconstruction. Because wavelet transforms work with large number of samples, the proposed architecture can be configured to have an arbitrary memory size (i.e. the powers of two) to cope with the application demands.

1.3 Thesis Outline

The rest of the thesis is organized as follows. The basic knowledge of wavelets are described in **Chap**. 2. This chapter also discusses the second generation of wavelet transforms, which is more popular under the name of lifting scheme. **Chap**. 3 discusses several

existing architectures to compute both DWT and DWP decomposition and reconstruction. Two different approaches are described here. It starts from the traditional wavelet transforms using filter banks, followed by the more recent lifting-based approaches. **Chap**. 4 details the algorithm and application mapping that maps the wavelet filters into lifting scheme in an efficient manner. Based on the mapping results, four different processing elements (PEs) are proposed. These PEs are the main block of the proposed wavelet processors. Additional components that make up the processors are also detailed here. The end of this chapter describes several transform processes and how they are supported and executed by our wavelet processors. **Chap**. 5 details the analysis of our wavelet processors, including the performances and the benchmarks. **Chap**. 6 concludes the thesis and shows possible future directions.

Chapter 2

Wavelets and Wavelet Algorithms

Contents

2.1 Time-Frequency Representation 5

2.1.1 The Heisenberg Uncertainty Principle 7

2.2 Short-Time Fourier Transform 9

2.2.1 Gabor Transform 10

2.2.2 Properties of Short-Time Fourier Transform 10

2.2.3 Discrete Representation of STFT 11

2.3 Continuous Wavelet Transform 12

2.4 Properties of Wavelets 13

2.4.1 Admissible Condition 14

2.4.2 Regularity 16

2.4.3 Localization Analysis 16

2.4.4 Wavelet Transform of Basic Functions 17

2.5 Time-Frequency Plane 17

2.6 Discrete Wavelet Transform 19

2.7 Multiresolution Signal Analysis 23

2.8 Two-Scale and Decomposition Relations 25

2.8.1 Relation between Two-Scale Sequences 28

2.8.2 Relation between Reconstruction and Decomposition Sequences . 29

2.9 Filter Banks 30

2.9.1 Decimation and Interpolation 31

2.9.2 Decomposition and Reconstruction Algorithms 32

2.9.3 Two-Channel Perfect Reconstruction Filter Bank 34

2.10 Polyphase Representation for Filter Banks 36

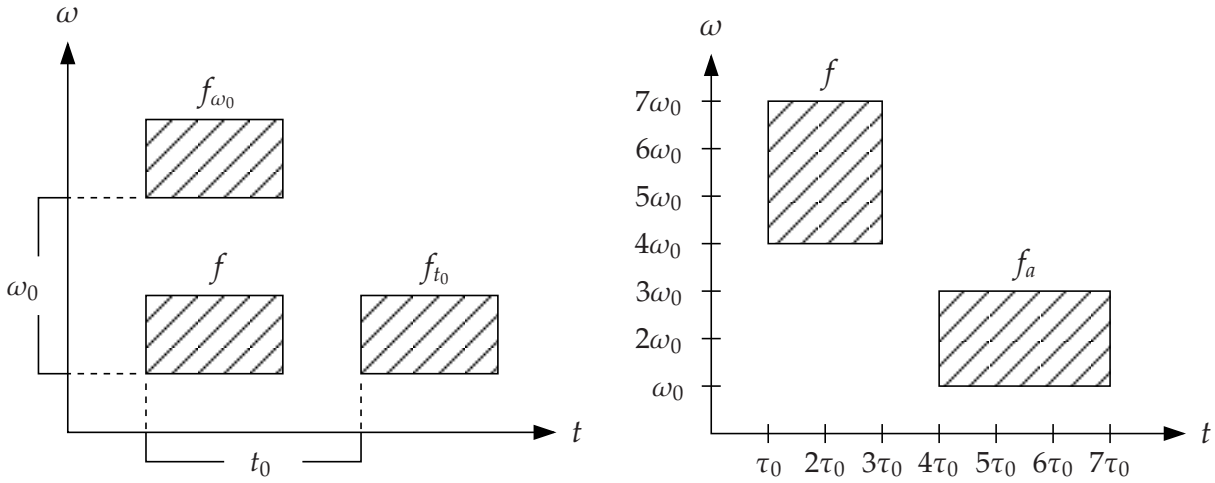
2.11 Lifting Scheme 37

This chapter describes the essential backgrounds of wavelets. Because Fourier analysis is a well known signal analysis tool, the discussion about wavelets is also presented using this approach. **Sec**. 2.1 describes the time-frequency representation of a signal and two elementary operations that influence the time-frequency plane. The solution of the Fourier analysis in time-frequency plane is detailed in **Sec**. 2.2. **Sec**. 2.3 starts the discussion about wavelets in the continuous-time domain and **Sec**. 2.4 describes their properties. Time-frequency plane of of the wavelets, along with several other bases, are detailed in **Sec**. 2.5. **Sec**. 2.6 continues the discussion in discrete-time domain. Multiresolution signal analysis of the wavelets is covered in **Sec**. 2.7 and relations to have a perfect reconstruction are described in **Sec**. 2.8. **Sec**. 2.9 details the methods to compute the wavelet transforms using filter banks and **Sec**. 2.10 details the polyphase representation of filter banks. **Sec**. 2.11 closes the discussion with lifting scheme. Some basic knowledge of Fourier analysis are explained in **App**. A.

2.1 Time-Frequency Representation

Time-frequency analysis is a substantial tool in the modern signal processing. Additional insight into the nature of signals can be achieved once the information about the distribution of the energy in those signals with respect to both time and frequency are extracted. Fourier series are an ideal tool to analyze periodic signals because the harmonic modes used in the Fourier expansions are themselves periodic. In contrary, the Fourier transform is a far less natural tool, because it uses periodic functions to represent the non periodic signals. Closely look to boundary condition of the Fourier transform, it is obvious that the transform can only be carried out when the entire signal in the whole of the real line $(-\infty, \infty)$ is known. Also, because the functions $e^{j\omega t}$ are *global functions* [?], a small change of the function at any point in the time domain influences every point on the frequency domain and vice versa. Another study on the Fourier transform is that the transform can be evaluated at only one frequency at a time. Albeit several algorithms exist to perform the transform in the digital domain, such as FFT, it is necessary to store the whole sampled data in the memory before the computation takes place.

Although Fourier analysis is a powerful tool in signal processing, it becomes inadequate when the local frequency contents of a signal are in the point of interest. Once the signal is transformed in the frequency domain with the Fourier analysis, the time information about the signal is lost. As an example, locations of the spikes that occur on the power line need to be determined. The Fourier analysis of this signal will give the 50/60 Hz as the major frequency in the frequency spectrum and contain infinite small number of sinusoidal functions as side lobes that make up the spikes (spikes can be represented as delta function which contains sharp changes). The more the spikes occur, the



(a) Shifting (b) Scaling

Fig. 2.1: Two elementary operations on a basis function f and their effects on the time-frequency plane [?].

higher the side lobes spectrum contributes. Nevertheless, since the 50/60 Hz contributes as the major existing frequency during the whole signal spans, the side lobes are still negligible. A local analysis is required to combine both time and frequency domain analyses to correct this imperfection, in order to achieve the *time-frequency analysis*. With this technique, the region of interest can be defined thus the local frequency contents of a signal can be extracted.

Another inadequate representation by the Fourier analysis is that when the signal expansion is about to be calculated, the primary concern of the expansion is the localization of a given basis function in frequency. Looking at its basis functions, the functions used in the Fourier analysis, $e^{j\omega t}$, are infinitely sharp in their frequency localization, which means that they exist at once precise frequency, but spread over infinite time. In other word, time localization ceases to exist after the analysis because of their infinite extent.

Classic examples that aim to capture the local frequency contents of a signal are the Short-Time Fourier Transform (STFT) [**?**] and Gabor transform [**?**]. These transforms use complex exponential window functions and their translations as expansion functions. Thus, it is no longer necessary to have the entire signal to perform the transform in order to extract the local frequency contents with some degree of accuracy.

The localization is related by the degree of spreading in both time and frequency domains. As an example, by defining the intervals I_t and I_ω, a *tile* in the time-frequency domain can be defined. In case of real basis functions, time-frequency representation would contain two mirror tiles at positive and negative frequencies. Thus, in order to simplify the illustration of the time-frequency plane, complex basis function is used as an assumption. **Fig**. 2.1 depicts this time-frequency illustration.

Consider two elementary operations *shifting* and *scaling* on a basis function f. Take a look at the shifting property of the Fourier transform, the shifting relation in time domain

can be written as

$$f_{t_0}(t) = f(t - t_0) \tag{2.1}$$

It is clear that a shifting of a function f in time by t_0 leads to a shifting of the tile in time axis by t_0. Likewise, a modulation by $e^{j\omega_0 t}$ on a function f leads to a shifting of the tile by ω_0 in frequency axis. **Fig**. 2.1(a) depicts the scheme of the effect of shifting on a time-frequency plane.

A function f scaled by a is written as

$$f_a(t) = f(at) \tag{2.2}$$

Following the scaling property of the Fourier transform, the scaling operation leads to

$$I'_t = \frac{1}{a} I_t \tag{2.3}$$

$$I'_\omega = a I_\omega \tag{2.4}$$

which alters both the shape and the localization of the tile, as depicted in **Fig**. 2.1(b). Note that all elementary operations conserve the surface of the time-frequency tile. In the case of scaling operation, the frequency resolution is traded for the resolution in time and vice versa.

2.1.1 The Heisenberg Uncertainty Principle

Two extreme cases can be drawn to illustrate the localization trade-off. By having a basis function $f(t) = e^{j\omega_0 t}$, its Fourier transform would contain exactly one frequency ω_0

$$f(t) = e^{j\omega_0 t} \longleftrightarrow F(\omega) = 2\pi\delta(\omega - \omega_0) \tag{2.5}$$

which tells that its frequency content is zero everywhere, except for $\omega = \omega_0$. It indicates a perfect localization in the frequency domain. Now if a basis function is a delta function $f = \delta(t - t_0)$, the transform pair is given by

$$f(t) = \delta(t - t_0) \longleftrightarrow F(\omega) = \frac{1}{2\pi} e^{-j\omega t_0} \tag{2.6}$$

The frequency content is nonzero on the whole ω-axis. This indicates a perfect localization in the time domain, but not in the frequency domain. It is impossible to get perfect localization in both time and frequency domains simultaneously, as detailed in [**?**, ?, **?**].

In general, the weighted means in the time domain and in the frequency domain can be considered as probability density functions (PDFs). Consider a function $f(t)$ with its corresponding Fourier transform $F(\omega)$ whose energy is centered around at (t_0, ω_0) in both

time and frequency domains

$$t_0 = \frac{\int t \left|f(t)\right|^2 dt}{\int \left|f(t)\right|^2 dt} \tag{2.7}$$

$$\omega_0 = \frac{\int \omega \left|F(\omega)\right|^2 d\omega}{\int \left|F(\omega)\right|^2 d\omega} \tag{2.8}$$

The standard deviations, which measure the width of the distribution of the time-frequency distribution of f around (t_0, ω_0), are defined as

$$s^2 = \frac{\int (t - t_0)^2 \left|f(t)\right|^2 dt}{\int \left|f(t)\right|^2 dt} \tag{2.9}$$

$$S^2 = \frac{\int (\omega - \omega_0)^2 \left|F(\omega)\right|^2 d\omega}{\int \left|F(\omega)\right|^2 d\omega} \tag{2.10}$$

These s and S are given as the square root of the variances and they define the required measurement for the *Heisenberg uncertainty principle*, which states

$$s^2 S^2 \geq \frac{1}{4} \tag{2.11}$$

This equation holds only for Gaussian signals

$$f(t) = e^{-\frac{(t-t_0)^2}{2\sigma^2}} \tag{2.12}$$

whose Fourier transform is another such Gaussian and the above width measure is related to the variance σ.

Another representation of the Heisenberg uncertainty principle, as found in different text books, is by its time width and frequency width:

$$\Delta_t^2 = \int t^2 \left|f(t)\right|^2 dt \tag{2.13}$$

$$\Delta_\omega^2 = \int \omega^2 \left|f(t)\right|^2 dt \tag{2.14}$$

If $f(t)$ vanishes faster than $1/\sqrt{t}$ as $t \to \pm\infty$, then

$$\Delta_t^2 \Delta_\omega^2 \geq \frac{\pi}{2} \tag{2.15}$$

Have a look at the textbooks [?, ?, ?] for the detail proof.

The uncertainty principle sets a relation on the maximum joint sharpness or resolution in time and frequency of any linear transform, which fundamentally states that precise measuring the moment of a particle makes the position of the particle uncertain, and conversely precise locating a particle in a small region of spaces makes the momentum of the particle uncertain. It is obvious that scaling only exchanges one resolution for the other and does not alter the time-bandwidth product, as depicted in **Fig**. 2.1.

2.2 Short-Time Fourier Transform

By defining a window on the Fourier transform, frequency contents of the region of interest of a signal can be extracted. The signal is multiplied by a window function before the Fourier transform is performed. This leads to the Short-Time Fourier Transform (STFT), defined as

$$G_w f(\tau, \xi) = \int_{-\infty}^{\infty} f(t)\, w^*_{\tau,\xi}(t)\, dt \tag{2.16}$$

with

$$w_{\tau,\xi}(t) = w(t-\tau)\, e^{j\xi t} \tag{2.17}$$

The window function $w(t)$ can be complex and satisfy the condition

$$W(0) = \int_{-\infty}^{\infty} w(t)\, dt \neq 0 \tag{2.18}$$

which states that $W(\omega)$ functions as a low-pass filter, i.e. nonzero spectrum at $\omega = 0$. STFT, also known as the *windowed Fourier transform* or *running-window Fourier transform*, can be computed after the signal interval in which $w(t-\tau)$ is available. The window function $w_{\tau,\xi}(t)$ itself can be considered as a *packet of waves* where the basis function $e^{j\xi t}$ oscillates inside the envelope function $w(t)$.

By taking the inverse Fourier transform of $G_w f(\tau, \xi)$, the time function $f_\tau(t)$ can be recovered. It is given by

$$f_\tau(t) = w(t-\tau) f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} G_w f(\tau, \xi) e^{j\xi t}\, d\xi \tag{2.19}$$

The original $f(t)$ is obtained by multiplying the recovered time function $f_\tau(t)$ with the window function $w^*(t-\tau)$ and integrating over τ.

$$f(t) = \frac{1}{2\pi \left\| w(t) \right\|} \int_{-\infty}^{\infty} d\xi e^{j\xi t} \int_{-\infty}^{\infty} G_w f(\tau, \xi) w^*(t-\tau) d\tau \tag{2.20}$$

where $\|w(t)\|$ is the norm of the window function $w(t)$, defined as

$$\begin{aligned} \left\| w(t) \right\| &= \sqrt{\langle w(t), w(t) \rangle} \\ &= \sqrt{\int_{t \in \mathbb{R}} |w(t)|^2\, dt} \end{aligned} \tag{2.21}$$

2.2.1 Gabor Transform

The Gabor transform was developed by D. Gabor [?]. The Gabor transform is a STFT with Gaussian function as the window function. The window function of the Gabor transform is defined as

$$g_\alpha(t) = \frac{1}{2\pi\alpha} e^{-\frac{t^2}{4\alpha}}, \qquad \alpha > 0 \tag{2.22}$$

Its Fourier transform is given by

$$G_\alpha(\omega) = e^{-\alpha\omega^2}, \qquad \alpha > 0 \tag{2.23}$$

By using **Eq**. (2.7)–**Eq**. (2.10), the window property of $g_\alpha(t)$ can be computed to give $t_0 = 0$, $\omega_0 = 0$, $s^2_{g_\alpha} = \alpha$, and $S^2_{G_\alpha} = \frac{1}{4\alpha}$. Thus, **Eq**. (2.11) becomes $s^2S^2 = \frac{1}{4}$, which achieves the lowest boundary of the Heisenberg uncertainty principle.

2.2.2 Properties of Short-Time Fourier Transform

Two fundamental properties of the STFT, namely *time shift* and *frequency shift*, are discussed as a tool to understand how to the effect of the STFT on the time-frequency plane.

Time Shift Let $f_{t_0}(t) = f(t - t_0)$ be a shifted version of a signal $f(t)$ by t_0. Its STFT is given by

$$\begin{aligned} G_w f_{t_0}(\tau, \xi) &= \int_{-\infty}^{\infty} f(t - t_0) w(t - \tau) e^{-j\xi t}\, dt \\ &= \int_{-\infty}^{\infty} f(t) w(t - (\tau - t_0)) e^{-j\xi t} e^{-j\xi t_0}\, dt \\ &= e^{-j\xi t_0} G_w f(\tau - t_0, \xi) \end{aligned} \tag{2.24}$$

This indicates that the location of STFT in the time-frequency plane will be shifted by the same amount t_0 in time axis while the frequency location will remain the same. The term $e^{-j\xi t_0}$ contributes to the phase change of the STFT, which is directly proportional to the time shift.

Frequency Shift Let f_{ω_0} be the modulated function of carrier signal $e^{j\omega_0 t}$ such that

$$f\omega_0(t) = f(t) e^{j\omega_0 t} \tag{2.25}$$

Its STFT is given by

$$\begin{aligned} G_w f_{\omega_0}(\tau, \xi) &= \int_{-\infty}^{\infty} f(t) e^{j\omega_0 t} w(t - \tau) e^{-j\xi t}\, dt \\ &= G_w f(\tau, \xi - \omega_0) \end{aligned} \tag{2.26}$$

This implies that the location of STFT in the time-frequency plane will be shifted by the same amount ω_0 along the frequency axis while both the magnitude and the phase remain unchanged.

Fig. 2.2 illustrates the effect of different time shifts (t_1, t_2, and t_3) and frequency shifts (ω_1, ω_2, and ω_3) of STFT in the time-frequency plane. It is obvious that the STFT does not alter the dimensions of the tile, it only moves the tile to the other location. The dimensions of the tile are fixed and determined by the root mean square (RMS) values of the

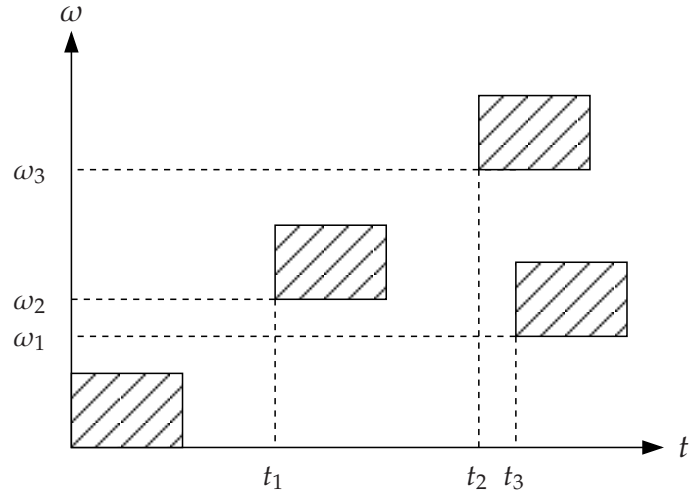


Fig. 2.2: Time shift and frequency shift operation on STFT and their effect on time-frequency plane [?].

window function used to localize the transform, as derived from **Eq**. (2.9) and **Eq**. (2.10). Practically, the time shift amount t_0 and the frequency modulation amount ω_0 are not arbitrarily chosen. Their values are based on the time and frequency intervals of the window function in order to avoid the unnecessary overlapping tiles on the time-frequency plane.

2.2.3 Discrete Representation of STFT

Recalling **Eq**. (2.17), the discrete form of the STFT is achieved by choosing $\omega = m\omega_0$ and $\tau = nt_0$. The corresponding window function becomes

$$w_{m,n}(t) = w(t - mt_0)e^{jn\omega_0 t} \tag{2.27}$$

with $m, n \in \mathbb{Z}$ and $\omega_0 > 0$ and $t_0 > 0$ have fixed values.

Provided that $w_{m,n}$ constitutes a *frame*, to reconstruct a given function f from its transform coefficients $(\langle w_{m,n}, f\rangle)_{m,n}$, the inverse formula of the discrete STFT becomes

$$\sum_{m,n} \langle w_{m,n}, f\rangle \, \tilde{w}_{m,n} = f = \sum_{m,n} \langle \tilde{w}_{m,n}, f\rangle \, w_{m,n} \tag{2.28}$$

where $\tilde{w}_{m,n}$ are the vectors of the dual frame and

$$G_w f(m,n) = \langle g_{m,n}, f\rangle = \int_{-\infty}^{\infty} f(t) w^*(t - nt_0) e^{-jm\omega_0 t} \, dt \tag{2.29}$$

2.3 Continuous Wavelet Transform

The STFT discussed previously offers advantages over the standard Fourier transform in extracting the local properties of a signal. In spite of that, the STFT poses a serious

problem in many applications because the time-frequency resolution of the STFT is fixed throughout the processing. Once the window function for STFT is defined, it is not possible to alter the time resolution or the frequency resolution to get a better observation of the signal. As an example, by choosing a window function that is sensitive to the low frequency portion, the resolution to extract the content of high frequency portion is very poor and vice versa.

Recalling from the Fourier transform, the Fourier coefficients of a function f are given by $c_n = \left\langle e^{jn\omega t}, f \right\rangle$. If the local frequency contents are at interest, the basis functions $e^{jn\omega t}$ need to be replaced by basis functions with a finite width on both time and frequency axis. The shifting operation on a finite width basis function extracts the local frequency information at time interval of interest, while the scaling operation on a basis function, which corresponds to contraction and stretching, defines the frequency resolution. These combined operations on a basis function are the principle of the wavelet transform.

The basis function, which is called the *mother* function $\psi(t)$, must satisfy

$$\int_{-\infty}^{\infty} \psi(t)\, dt = \Psi(0) = 0 \tag{2.30}$$

to achieve band-pass filter with impulse response and zero mean. Here, a two-parameter family of expansion functions $\psi\left(\frac{t-b}{a}\right)$ is observed to cover both time and frequency resolutions. The basis function is normalized to conserve energy, given by

$$\left\|\psi(t)\right\|^2 = 1 \tag{2.31}$$

It is apparent that

$$\left\|\psi\left(\frac{t-b}{a}\right)\right\|^2 = a \tag{2.32}$$

The factor $\frac{1}{\sqrt{a}}$ is introduced to conserve the energy on the expansion functions. Thus, the expansion functions will be in form

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{2.33}$$

The Continuous Wavelet Transform (CWT) is given by

$$\mathscr{W}_\psi f(a,b) = \frac{1}{\sqrt{a}} \int_{\mathbb{R}} \psi^*\left(\frac{t-b}{a}\right) f(t)\, dt \tag{2.34}$$

where $a \in \mathbb{R}^+$ and $b \in \mathbb{R}$. This transform measures the similarity between the signal f and shifts and scales of a basis function, which can be rewritten as

$$\mathscr{W}_\psi f(a,b) = \left\langle \psi_{a,b}(t), f(t) \right\rangle \tag{2.35}$$

and inverse wavelet transform is defined as

$$f(t) = \mathscr{W}_\psi^{-1} F(t) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(a,b)\psi_{a,b}(t)\, db\, da \tag{2.36}$$

The behaviour of a wavelet can be observed using by frequency analysis tools. Given is a mother function of a wavelet $\psi(t)$, the Fourier transform of the wavelet is

$$\begin{aligned}\Psi_{a,b}(\omega) &= \int \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) e^{-j\omega t}\, dt \\ &= \sqrt{a}\, \Psi(a\omega)\, e^{-j\omega b}\end{aligned} \tag{2.37}$$

with $\Psi(\omega)$ as the Fourier transform of the mother wavelet $\psi(t)$. In the frequency domain, the wavelet is normalized by a factor $\sqrt{a}$, scaled by $1/a$, and multiplied by a phase factor $e^{-j\omega b}$. This shows a well known concept of scaling, where the parameter a determines the scaling parameter, which compresses or stretches the basis function on the time axis. Therefore, the expansion functions $\psi_{a,b}(t)$ will have changing time-frequency tiles. For large a ($a > 1$), the tile will be long in time axis and low in frequency axis, while for small a ($a < 1$), the tile will be short in time axis and high in frequency axis.

The correlation between the function f and the wavelet ψ in the time domain can be written as the inverse Fourier transform of the product of the conjugate Fourier transform of the wavelet and the Fourier transform of the function. It is expressed as

$$\mathscr{W}_\psi f(a,b) = \frac{\sqrt{a}}{2\pi} \int F(\omega)\Psi^*(a\omega)e^{j\omega b}\, d\omega \tag{2.38}$$

The Fourier transform of the wavelets has shown also how the wavelet transform is performed. The term $\sqrt{a}\Psi(a\omega)$ in **Eq**. (2.37) corresponds to the impulse response of the scaled wavelet transform filter $\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)$. This means that the wavelet transform is a multi-bank wavelet transform filters with different scales of a.

2.4 Properties of Wavelets

The difference between the wavelet transform and other transforms such as Fourier transform is that in the wavelet transform, the mother wavelet $\psi(t)$, which is the kernel function, is not specified. Nevertheless, the basis function that builds up the wavelet transform must satisfy the basic conditions such as admissibility, regularity, and orthogonality to make the wavelet transform work. Thus, talking about wavelet transform must also specify what wavelet is used in the transform. Before discussing the conditions that need to be satisfied for a mother wavelet, some general properties of wavelet transforms are briefly discussed here.

Linearity Similar to the Fourier transform, the wavelet transform inherits the linearity principle of the inner product

$$\mathscr{W}_\psi(\alpha f + \beta g) = \alpha(\mathscr{W}_\psi f)(a,b) + \beta(\mathscr{W}_\psi f)(a,b) \tag{2.39}$$

Time Invariant If f_u represents the shifted function of f, denoted as $f_u = f(t-u)$, its wavelet transform is also shifted.

$$\begin{aligned}\mathscr{W}_\psi f_u(a,b) &= \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty}\psi_{a,b}\left(\frac{t-b}{a}\right)f(t-u)\,dt \\ &= \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty}\psi_{a,b}\left(\frac{\tau+u-b}{a}\right)f(\tau)\,d\tau \\ &= \mathscr{W}_\psi f(a,b-u)\end{aligned} \tag{2.40}$$

Dilation Given is

$$f_v(t) = \sqrt{v}f(vt) \tag{2.41}$$

with $v \in \mathbb{R}^+$, the corresponding wavelet transform is

$$\begin{aligned}\mathscr{W}_\psi f_v(a,b) &= \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty}\psi\left(\frac{t-b}{a}\right)\sqrt{v}f(vt)\,dt \\ &= \frac{1}{\sqrt{a}}\frac{1}{\sqrt{v}}\int_{-\infty}^{\infty}\psi_{a,b}\left(\frac{\tau/v-b}{a}\right)f(\tau)\,d\tau \\ &= \mathscr{W}_\psi f\left(\frac{a}{v},vb\right)\end{aligned} \tag{2.42}$$

The rest of this section discusses some basic conditions that must be satisfied by a kernel function. One of them is related to the fact that the wavelet transform must be invertible, which means that the original signal can be perfectly reconstructed after the transform. To satisfy this condition, properties that the mother wavelet must hold, such as the resolution of the identity, the energy conservation in the time-frequency domain, and the wavelet admissible condition, will be discussed. The fact that the wavelet transform should be a local operation in both time and frequency domains is related by the regularity condition. Finally, properties regarding the localization in time and frequency domains and examples of wavelet transforms of some basic functions are covered. The readers are suggested to look at the textbooks [**?**, **?**, **?**, **?**, ?] for complete explanations.

2.4.1 Admissible Condition

The wavelet transform transforms a one-dimensional signal into a two-dimensional time-frequency joint representation. During the wavelet transform, information regarding the signal must be preserved. Hence, the resolution of the identity must be satisfied, expressed as

$$\int\frac{da}{a^2}\int db\left\langle f_1,\psi_{a,b}\right\rangle\left\langle\psi_{a,b},f_2\right\rangle = c_\psi\left\langle f_1,f_2\right\rangle \tag{2.43}$$

where c_ψ is a constant. The term $1/a^2$ is the contribution of the time-frequency differential elements, that is $db\,d(1/a) = db\,da/a^2$. In the Fourier domain, the resolution of the identity

can be written as

$$\begin{aligned}
&\int \frac{da}{a^2} \int db \left\langle f_1, \psi_{a,b} \right\rangle \left\langle \psi_{a,b}, f_2 \right\rangle \\
&= \frac{1}{4\pi^2} \int \frac{da}{a^2} \int db \iint a F_1(\omega_1) \Psi^*(a\omega_1) F_2^*(\omega_2) \Psi(a\omega_2) e^{jb(\omega_1-\omega_2)} \, d\omega_1 \, d\omega_2 \\
&= \frac{1}{2\pi} \iint F_1(\omega_1) F_2^*(\omega_1) \left|\Psi(a\omega_1)\right|^2 \frac{da}{a} \, d\omega_1
\end{aligned} \tag{2.44}$$

By substituting $\omega = a\omega_1$ and $da = \frac{d\omega}{|\omega_1|}$ so that da and $d\omega$ are of the same sign, which makes $\frac{da}{a} = \frac{d\omega}{|\omega|}$, and defining

$$c_\psi = \int \left|\Psi(\omega)\right|^2 \frac{d\omega}{\omega} \tag{2.45}$$

Eq. (2.44) is simplified to

$$\int \frac{da}{a^2} \int db \left\langle f_1, \psi_{a,b} \right\rangle \left\langle \psi_{a,b}, f_2 \right\rangle = \frac{c_\psi}{2\pi} \int F_1(\omega_1) F_2^*(\omega_1) \, d\omega_1 \tag{2.46}$$

Following the Parseval's equality, it leads to

$$\frac{1}{2\pi} \int F_1(\omega_1) F_2^*(\omega_1) \, d\omega_1 = \int f_1(t) f_2^*(t) \, dt = \left\langle f_1, f_2 \right\rangle \tag{2.47}$$

Hence, the condition of the resolution of the identity is satisfied when

$$c_\psi = \int \frac{\left|\Psi(\omega)\right|^2}{|\omega|} < +\infty \tag{2.48}$$

The equation above implies the admissible condition of the wavelet, that is the Fourier transform of the wavelet must be equal to zero at the zero frequency

$$\left|\Psi(\omega)\right|\Big|_{\omega=0} = 0 \tag{2.49}$$

or in the time domain

$$\int_{-\infty}^{\infty} \psi(t) \, dt = \Psi(0) = 0 \tag{2.50}$$

which equals to **Eq**. (2.30) as mentioned at the beginning. The admissible condition describes that the wavelet must oscillate to have its mean value equal to zero.

Recalling **Eq**. (2.43), by setting $f = f_1 = f_2$, the resolution of the identity becomes

$$\iint \left|\mathcal{W}_\psi f(a,b)\right|^2 db \frac{da}{a^2} = c_\psi \int \left|f(t)\right|^2 dt \tag{2.51}$$

which is the energy conservation relation of the wavelet transform. This formula is equivalent to the Parseval's energy relation in the Fourier transform from **Eq**. (A.20).

2.4.2 Regularity

Regularity is not an essential condition to design the wavelet, but it is usually required to satisfy the property of the wavelet, that is the wavelet transform should be local in both time and frequency domains.

Recapitulating from the moment property of the Fourier transform, the nth-order moment of the wavelet is expressed as

$$m_n = \int_{-\infty}^{\infty} t^n \psi(t)\, dt \tag{2.52}$$

From the admissible condition of the wavelet, it is clear that $m_0 = 0$. In general, it is required to have a wavelet whose its first $n + 1$ moments equal to zero. That is

$$m_p = \int_{-\infty}^{\infty} t^p \psi(t)\, dt = 0 \qquad for\ p = 0, 1, 2, .., n \tag{2.53}$$

The wavelet that satisfies this condition is called the wavelet of order n. This equation describes that the wavelet must have an exponential decay so that its first low order moments are equal to zero and the order $n + 1$ measures the flatness of the wavelet in the frequency domain about $\omega = 0$.

The regularity plays an important role in signal compression. Consider the following case: Wavelet transform of a one-dimensional signal is two-dimensional, and wavelet transform of a two-dimensional signal is four-dimensional. This leads to an explosion of the time-bandwidth product with the wavelet transform, which is in contradiction with the goal of the signal compression. For this purpose, the regularity condition guarantees the wavelet transform coefficients decrease rapidly with decreasing of the scale a and increase with increasing of $1/a$.

2.4.3 Localization Analysis

The goal of time-frequency signal analysis is to decompose a signal into multiple frequency bands, so that each decomposed band can be processed in a different and independent manner. For this purpose, the wavelet operator must be local in both time and frequency domains. Historically, it has been a difficult research topic to find wavelets that satisfy this condition.

By investigating both mentioned conditions, the wavelet must oscillate to have a zero mean and decay as fast as t^{-n} for the wavelet of order n. This implies that the wavelet must be a small wave that oscillates and vanishes, which gives the property of localization in time domain of the wavelet transform. In the frequency domain, according to regularity condition and **Eq**. (2.37), the wavelet must decay rapidly with the frequency ω, which gives the property of localization in frequency domain. Finally, from **Eq**. (2.30) or **Eq**. (2.50), the wavelet must be equal to zero at the zero frequency, which makes the wavelet acts as a band-pass filter.

2.4.4 Wavelet Transform of Basic Functions

Because the mother function of a wavelet is not clearly specified, the wavelet transform is not ready for closed form solution. Nevertheless, some simple functions can be deduced analytically to give an insight view about the wavelet transforms.

Constant Signal Given $f(t) = c$, from the admissible condition of the wavelet defined by **Eq**. (2.50), its transform is

$$\mathscr{W}_\psi f(a,b) = 0 \tag{2.54}$$

Sinusoidal Function For a sinusoidal function, expressed as $f(t) = e^{j\omega_0 t}$, its wavelet transform is a direct solution of the Fourier transform of the wavelet in **Eq**. (2.38)

$$\mathscr{W}_\psi f(a,b) = \sqrt{a}\Psi^*(a\omega_0)e^{j\omega_0 b} \tag{2.55}$$

which says that the wavelet transform of a sinusoidal function is a sinusoidal function of the time shift b.

Linear Function For a linear function $f(t) = t$, its wavelet transform is

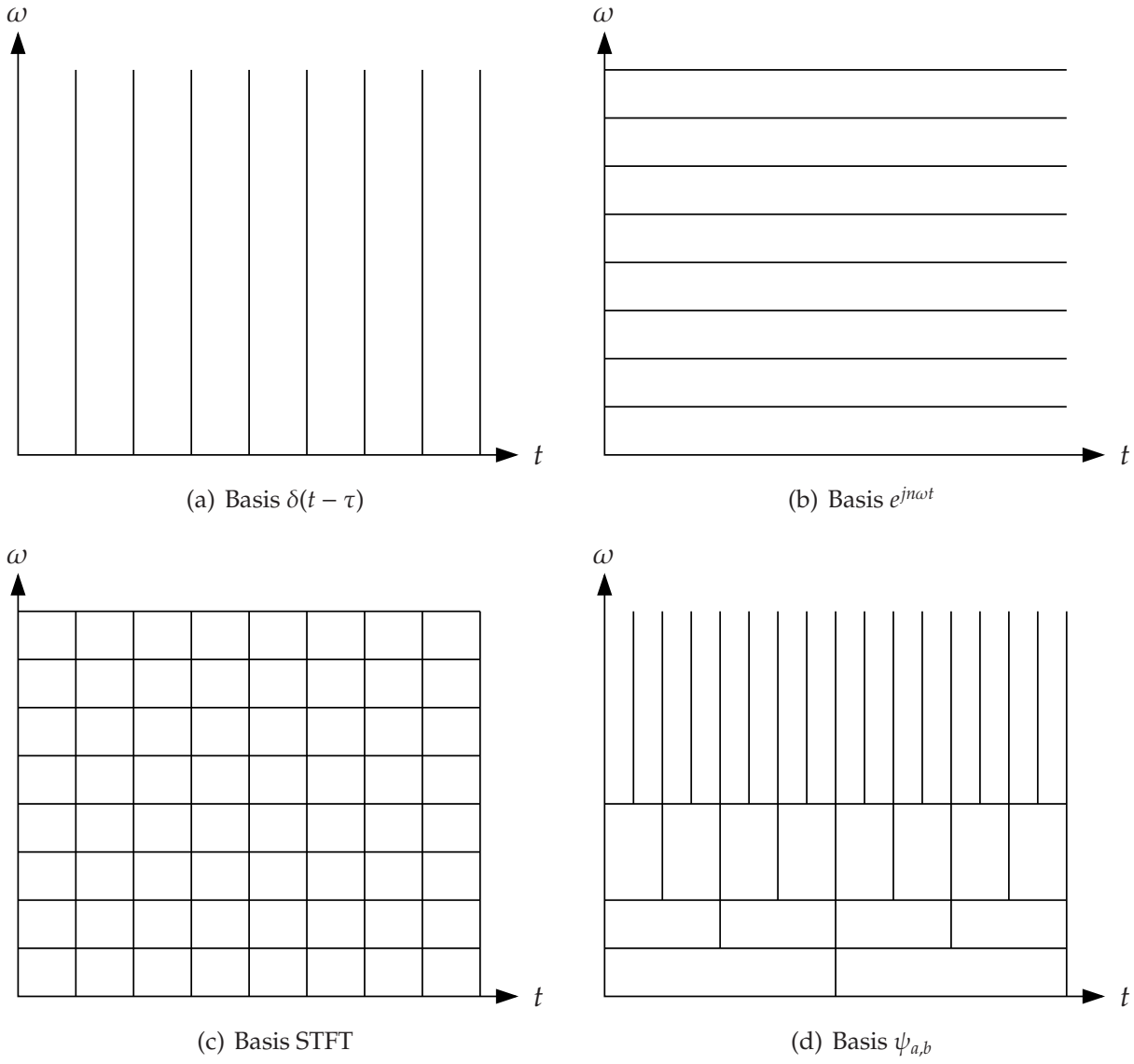
$$\begin{aligned}\mathscr{W}_\psi f(a,b) &= \frac{1}{\sqrt{a}}\int t\psi^*\left(\frac{t-b}{a}\right)dt \\ &= a^{3/2}\int t\psi^*(t-b')\,dt \\ &= \frac{a^{3/2}}{i}\frac{d\Psi^*(\omega)}{d\omega}\Big|_{\omega=0}\end{aligned} \tag{2.56}$$

If the wavelet follows the regularity property and is at least of order one, its first derivative of first-order is equal to zero at $\omega = 0$. Thus the wavelet transform of a linear function $f(t) = t$ is also equal to zero.

2.5 Time-Frequency Plane

The representation of linear expansion of signals or functions is a change of basis. Determining a good basis for the expansion is the main challenge, because it depends not only on the class of the signals that will be represented, but also on the criterion of the quality of the expansion. In general, a good basis should deliver a compact representation of the signals and less complex processing. As examples, Fourier basis is suitable for the implementation of convolution and Karhunen-Loève is good for compression because it concentrates as much energy in as few coefficients as possible.

In case of multiresolution signal analysis, the linear expansions with some structure are at interest. These expansions have basis vectors that are related to each other by elementary operations such as shifting in time, shifting in frequency (modulation), and



(a) Basis $\delta(t-\tau)$ (b) Basis $e^{jn\omega t}$

(c) Basis STFT (d) Basis $\psi_{a,b}$

Fig. 2.3: Time-frequency plane of different bases [?, ?, ?].

scaling. Using these operations, resolution trade-offs between time versus frequency is permitted. Bases without such structure are useless for complexity reason if expansions for high-dimensional spaces are concerned. An elegant interpretation of such expansions was originated by Gabor [?]. It is drawn in terms of their time-frequency tiling where each basis function has a region in time-frequency plane where most of its energy is concentrated. This elementary tile on the time-frequency plane, originally called *logon*, corresponds to the value of the expansion coefficient.

Fig. 2.3 depicts the time-frequency plane of four different bases. Discrete-time series expansions, also often called discrete-time transforms, are used to give a clear illustration on the time-frequency tiles. The ideal expectation of having expansions is to capture both the isolated impulse and the isolated frequency component, in other words, which exact frequency component is composed by the signal at which exact time. The

first case, depicted in **Fig**. 2.3(a), shows schematically the tiling of identity basis $\delta(t-\tau)$. It is obvious that by having dirac in time $\delta(t-\tau)$ as a basis, the time information about signal is isolated but the frequency information about the signal is lost. Similar case is depicted in **Fig**. 2.3(b) where the Fourier basis $e^{jn\omega t}$ is used. The linear expansion with Fourier basis extracts the frequency contents of the signal exactly. Because Fourier basis is a global function, time information about the signal is lost in this case. **Fig**. 2.3(c) shows the case of STFT where the Fourier transform is applied on a windowed signal. Shifting in time is performed in a way that complete set of representation is extracted without any redundant or overcomplete representation. Because the window function is fixed, the resulting tiles also have fixed dimension. The local discrete-time Fourier series achieves a compromised solution by locating both time and frequency information about signal to a certain degree. The last time-frequency plane depicted in **Fig**. 2.3(d) shows the tiling with wavelet basis $\psi_{a,b}$. Wavelet basis uses shifting and scaling operations for the expansion. The scaling operation on wavelet basis contributes to the different dimension of the resulting tiles. Note again that discrete form is illustrated here for the sake of clearness. In the continuous domain, the tilings will have no boundaries. Additionally, the scaling factor of two is used in representing time-frequency plane on the wavelet basis. That is why the trade-offs between time and frequency about the signal have regular structures that follow the Heisenberg Uncertainty Principle.

2.6 Discrete Wavelet Transform

The continuous wavelet transform provides a tool that maps a one-dimensional time signal to a two-dimensional time-frequency joint representation. As the result of the transform, the output of the time-bandwidth product is the square of that of the original time signal. In contrary, the goal of signal processing for most applications is to represent the signal in a more efficient manner with as few parameters as possible. Thus, CWT is impractical for this purpose. Discrete Wavelet Transform (DWT), on the other side, can reduce the time-bandwidth product of the output of the wavelet transform.

DWT is in fact continuous wavelets with discrete scaling factor and discrete translation factor. Thus, the wavelet transform is analyzed at discrete scales and discrete translations.

Rewriting the wavelet expansion functions from **Eq**. (2.33),

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{2.57}$$

In order to reconstruct the original signal, the resolution of the identity formula, which is based on a double integral, is required in case of a CWT.

The discrete scaling factor of DWT is formulated as:

$$a = a_0^m \tag{2.58}$$

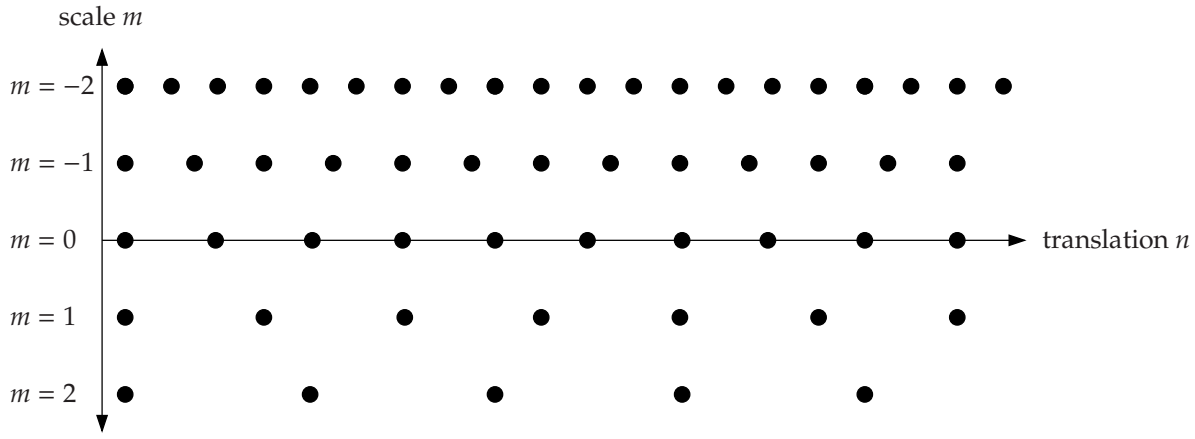


Fig. 2.4: Lattice structure of localization of the discrete wavelets in time-frequence plane [?, ?].

where m is integer and $a_0 > 1$ is a fixed scaling step. The discrete translation factor is formulated in such a way that $\psi(t - nb_0)$ will cover the whole discrete time axis. Since the basis functions are rescaled, the translation step b at scale m also depends on m. Similar to **Eq**. (2.13), by defining the width of the wavelet function, we have

$$\Delta_t \left(\psi_{a_0^m,0}(t) \right) = a_0^m \Delta_t \left(\psi(t) \right) \tag{2.59}$$

It is obvious that for $\psi_{a,b}(t)$ to cover the whole axis at a scale given by **Eq**. (2.58), the discrete translation factor has to be formulated as:

$$b = nb_0 a_0^m \tag{2.60}$$

where n is integer and b_0 is a translation step. The last term on the translation factor indicates that the translation depends on the scaling step a_0^m. The corresponding DWT is written as:

$$\begin{aligned} \psi_{m,n}(t) &= \frac{1}{\sqrt{a_0^m}} \psi \left(\frac{t - nb_0 a_0^m}{a_0^m} \right) \\ &= a_0^{-m/2} \psi \left(a_0^{-m} t - nb_0 \right) \end{aligned} \tag{2.61}$$

Discrete times and discrete scales that correspond to a sampling in the time-frequency space are used to evaluate the DWT. As the result, the time-frequency representation of a DWT is a tile along the time and frequency axes, as illustrated in **Fig**. 2.3(d). For the computer implementation, the DWT with the dyadic scaling factor $a_0 = 2$ can be implemented in an efficient way because orthonormal bases exist and reconstruction from the transform coefficients is possible. This will be discussed later in depth.

Fig. 2.4 depicts a better representation of the time-frequency distribution of the discrete wavelets in lattice form. For the illustration purpose, the scaling step a_0 is set to $2^{1/2}$ and translation step b_0 is set to 1. Different values of m correspond to wavelets of different width. For small values, high-frequency wavelets are translated by smaller steps which

make the distance between points shorter. For large values of m, low-frequency wavelets are translated by larger steps which make the distance between points larger. The scale axis m is in fact in form of a logarithmic scale if the frequency distribution is at interest.

Recalling **Eq**. (2.60), the sampling along the time axis is proportional to the scale a_0^m. This indicates that for small-scale wavelet analysis, the time sampling step is small, and for large-scale wavelet analysis, the time sampling step is large as well. With the varying scale, wavelet analysis can extract the details of the signal by using more concentrated wavelets of very small scale. For this detailed analysis of the signal, the time sampling step is also very small. Because only the signal detail is of interest, only a few small time translation steps will be needed. Therefore, the wavelet analysis provides a better solution to represent transient signals.

The parameter m plays an important role in the wavelet analysis. If very small details are at interest, the scaling must be large and this corresponds to a large and negative m. Also, with a large and negative m, according to **Eq**. (2.60), the step of translation is small. Thus, every detail will not be missed by the wavelet analysis. If a coarse feature of the signal is at interest, wavelet must be spread out and the large translation step will be adjusted to the width of the wavelet analysis function as well. This is accomplished by choosing a large and positive value for m.

In the continuous-parameter wavelet transform, the resolution of the identity from **Eq**. (2.36) is applied in order to reconstruct the signal from the given transform coefficients $\langle \psi_{a,b}, f \rangle$. In the discrete-parameter domain, no equivalent formula exists to reconstruct the signal. However, for certain ψ and appropriate values of a_0 and b_0, there exist $\tilde{\psi}_{m,n}$ which make the reconstruction possible. Thus, the function f can be reconstructed as follows:

$$f = \sum_m \sum_n \left\langle \psi_{m,n}, f \right\rangle \tilde{\psi}_{m,n} \tag{2.62}$$

Obviously, if a_0 is set close to one and b_0 is set close to zero, the double sum will become a close approximation to the double integral used in the resolution of the identity, thus the reconstruction of a function f should be possible. The only condition that needs to be satisfied is the admissible condition on the wavelet $\psi(t)$. This approximation approach to the ideal case is however not applicable when dealing with ordinary signal for real where the sampling is sparse, e.g. a_0 equals to two and b_0 is set to one. The reconstruction with **Eq**. (2.62) can be achieved only for special choices of the wavelets $\psi(t)$.

The conditions that must be satisfied in order to have a stable reconstruction need to be established. Given a function $f(t) \in L_2(\mathbb{R})$, the sum of the absolute square of its wavelet coefficients, that is $\sum_{m,n} |\langle \psi_{m,n}, f \rangle|^2$, has to be finite. This means that the operator that maps a function $f(t)$ into coefficients $\langle \psi_{m,n}, f \rangle$ has to be bounded. Additionally, no $f(t)$ with $\|f\| > 0$ should be mapped to 0. If these two conditions are satisfied, the stable reconstruction of a function $f(t)$ is guaranteed. These conditions lead to *frame bounds* which will be detailed shortly.

The first condition tells us that for any wavelet with some decay in time and also in

frequency (regularity property of wavelet), having zero mean (admissible condition), and any choice for $a_0 > 1$ and $b_0 > 0$, it can be shown that

$$\sum_{m,n} \left|\left\langle \psi_{m,n}, f \right\rangle\right|^2 \leq B\left\|f\right\|^2 \tag{2.63}$$

which states that the sequences $(\langle \psi_{m,n}, f \rangle)_{m,n}$ is in $\ell_2(\mathbb{Z}^2)$. In other words, the sequence is square-summable [?]. On the other hand, in order to guarantee the stable reconstruction, it is required that if $\sum_{m,n} |\langle \psi_{m,n}, f \rangle|^2$ is small, $\|f\|^2$ should be small as well. It tells us that $\sum_{m,n} |\langle \psi_{m,n}, f \rangle|^2$ should be close to $\|f\|^2$. This also means that there should exist $a < \infty$ such that

$$\sum_{m,n} \left|\left\langle \psi_{m,n}, f \right\rangle\right|^2 < 1 \tag{2.64}$$

which implies that $\|f\|^2 < \alpha$.

Given an arbitrary function f, with $\tilde{f}$ defined as

$$\tilde{f} = \left[\sum_{m,n} \left|\left\langle \psi_{m,n}, f \right\rangle\right|^2\right]^{1/2} f \tag{2.65}$$

It is obvious that

$$\sum_{m,n} \left|\left\langle \psi_{m,n}, \tilde{f} \right\rangle\right|^2 \leq 1 \tag{2.66}$$

which also implies that $\|\tilde{f}\|^2 \leq \alpha$. This corresponds to

$$A\left\|f\right\|^2 \leq \sum_{m,n} \left|\left\langle \psi_{m,n}, f \right\rangle\right|^2 \tag{2.67}$$

with some $A = 1/\alpha$. In addition, by substituting $f = f_1 - f_2$, **Eq**. (2.67) also indicates that if $\sum_{m,n} |\langle \psi_{m,n}, f_1 \rangle \langle \psi_{m,n}, f_2 \rangle|^2$ is small, the distance $\|f_1 - f_2\|$ cannot be arbitrarily large, which is equivalent to the stability requirement.

The theory of wavelet frames provides a general framework that allows us to balance between the redundancy, that is the density of the sampling, and the restriction on the wavelet $\psi(t)$. If the redundancy is small with critical sampling, then the wavelet bases are very constrained. On the other side, if the redundancy is large with high over-sampling, then the wavelet bases have only mild restrictions. Putting **Eq**. (2.63) and **Eq**. (2.67) all together sets a boundary that a numerically stable reconstruction of a function f from its wavelet coefficients is possible only if

$$A\left\|f\right\|^2 \leq \sum_{m,n} \left|\left\langle \psi_{m,n}, f \right\rangle\right|^2 \leq B\left\|f\right\|^2 \tag{2.68}$$

with $A > 0$, $B < \infty$, and A, B are independent of $f(t)$. This relation was proven by Daubechies in [?, ?]. If this condition is satisfied, then the family $(\psi_{m,n})_{m,n\in\mathbb{Z}}$ constitutes a *frame* with A, B as the frame bounds. Hence, when the relation between the energy of

the function and the energy of its discrete transform function is bounded between some value greater than zero and less than infinity for all possible square integrable functions, the completeness of the transform is guaranteed and no information is lost. Thus, the signal can be reconstructed from its decomposition wavelet coefficients.

Daubechies has shown that the accuracy of the reconstruction is influenced by the frame bounds A and B. These frame bounds can be computed from m, n, and the wavelet basis, $\psi(t)$. In order to have a more accurate reconstruction, it is necessary that the values of the frame bounds A and B are kept close. When $A = B$, the energy of the wavelet transform is proportional to the energy of the signal, which is similar to the energy conservation relation of the continuous wavelet transform from **Eq**. (2.51). Special case occurs when $A = B = 1$ and $|\psi_{m,n}| = 1$ for all m, n. This condition leads to a so-called *tight frame* where the family of the wavelets is an orthonormal basis. Here, the energy conservation is equivalent to the Parseval's relation of the Fourier transform.

It is important to note that the reconstruction of a signal f from its wavelet coefficients can be done even when the wavelets are not orthogonal to each other. In case when $A \neq B$, proportional relation between two energies can still be drawn. In order to reconstruct the original signal f from the decomposed wavelet coefficients, synthesis function basis is used. This basis is different from the decomposition function basis that is used to decompose the signal f. The synthesis function basis constitutes the dual frame of the decomposition function basis.

2.7 Multiresolution Signal Analysis

Multiresolution signal analysis (MRA) forms the most important building block for the reconstruction of *scaling functions* and wavelets. The idea of MRA was developed by Meyer [?] and Mallat [**?**, **?**]. Multiresolution signal processing divides a signal into different resolution. Resolution in this term defines how details a signal is represented at any given level of scale. The purpose of MRA is to decompose the signal in multiple frequency bands. Thus, the signal in each band can be studied separately and also differently.

Famous example of multiresolution signal processing tools is wavelet transform. Time-frequency plane in **Fig**. 2.3(d) clearly shows how the time information about signal is traded with the frequency information about the signal when the wavelet is used as the basis. The more precise measurement in time domain will result in the loss information in frequency domain and vice versa. Although the concept of multiresolution is general, the rest of this section will deal with multiresolution from wavelets theory point of view. More specific, the discrete form of the wavelet transform will be used as a starting point.

Multiresolution can be described as a set of subspaces $\{\mathbf{V_j}\}$ and $\{\mathbf{W_j}\}$. The subspaces $\{\mathbf{V_j}\}$ represent the set of the scaling spaces which contain the coarse information of the

signal and they are generated by the bases

$$\left\{\phi_{j,k} : 2^{j/2}\phi\left(2^j t - k\right);\ j,k \in \mathbb{Z}\right\} \tag{2.69}$$

The subspaces $\{\mathbf{W_j}\}$ represent the details of the signal which are extracted from the scaling spaces and they are generated by the bases

$$\left\{\psi_{j,k} : 2^{j/2}\psi\left(2^j t - k\right);\ j,k \in \mathbb{Z}\right\} \tag{2.70}$$

The wavelet space $\mathbf{W_j}$ contains the difference between $\mathbf{V_j}$ and $\mathbf{V_{j+1}}$. The sum of scaling space $\mathbf{V_j}$ and wavelet space $\mathbf{W_j}$ will generate a higher resolution scaling space $\mathbf{V_{j+1}}$. Thus it leads to

$$\mathbf{V_j} \cap \mathbf{W_j} = \{0\} \tag{2.71}$$

$$\mathbf{V_j} \oplus \mathbf{W_j} = \mathbf{V_{j+1}} \tag{2.72}$$

To achieve a MRA of a function, it is required to have a *scaling function* $\phi(t)$ that has a finite energy function. The scaling function does not satisfy the admissible condition, which means that the mean value of the scaling function $\phi(t)$ is not equal to zero and it is usually normalized to unity.

The scaling function generates a subspaces or better said, a nested sequence $\{\mathbf{V_j}\}$, namely

$$\{0\} \longleftarrow \cdots \subset \mathbf{V_{-1}} \subset \mathbf{V_0} \subset \mathbf{V_1} \subset \cdots \longrightarrow L^2 \tag{2.73}$$

and satisfies a dilation equation

$$\phi(t) = \sum_n h[n]\phi(mt - n) \tag{2.74}$$

with $\{h[n]\} \in \ell^2$ and $m > 0$. In case of a discrete form, $m = 2$, which corresponds to octave scales, is chosen. **Eq**. (2.74) is part of the two-scale relations that will be discussed later in **Sec**. 2.8. The function $\phi(t)$ is called scaling function, because it is represented as a superposition of a scaled and translated version of itself. Consequently, the following two properties hold for MRA

$$f(t) \in \mathbf{V_j} \Leftrightarrow f(2t) \in \mathbf{V_{j+1}} \tag{2.75}$$

$$f(t) \in \mathbf{V_j} \Leftrightarrow f(t + 2^{-j}) \in \mathbf{V_j} \tag{2.76}$$

These two properties are unique to MRA.

In short, any function $f_j(t) \in \mathbf{V_j}$ can be written as

$$f_j(t) = \sum_k v_{j,k}\phi\left(2^j t - k\right) \tag{2.77}$$

and any function $g_j(t) \in \mathbf{W_j}$ can be written as

$$g_j(t) = \sum_k w_{j,k}\psi\left(2^j t - k\right) \tag{2.78}$$

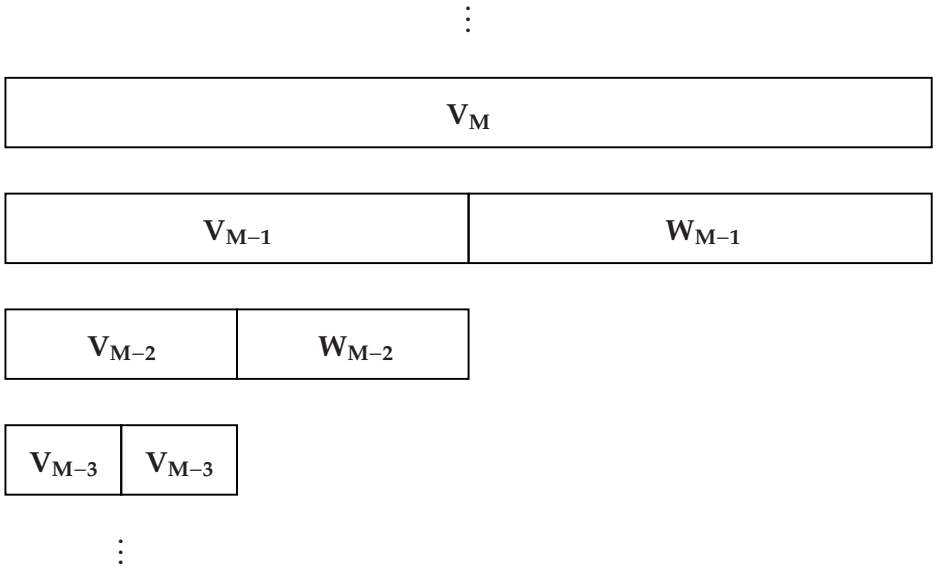


Fig. 2.5: Hierarchical structure of MRA subspaces [?].

for some coefficients $\{v_{j,k}\}_{j,k\in\mathbb{Z}}, \{w_{j,k}\}_{j,k\in\mathbb{Z}} \in \ell^2$.

Expanding **Eq**. (2.72),

$$\begin{aligned}\mathbf{V_{j+1}} &= \mathbf{W_j} \oplus \mathbf{V_j} \\ &= \mathbf{W_j} \oplus \mathbf{W_{j-1}} \oplus \mathbf{V_{j-1}} \\ &= \mathbf{W_j} \oplus \mathbf{W_{j-1}} \oplus \mathbf{W_{j-2}} \oplus \cdots \end{aligned} \tag{2.79}$$

we have

$$\mathbf{V_j} = \bigoplus_{\ell=-\infty}^{s-1} \mathbf{W}_\ell \tag{2.80}$$

Fig. 2.5 depicts a schematic representation of a hierarchical nature of MRA of $\mathbf{V_j}$ and $\mathbf{W_j}$.

2.8 Two-Scale and Decomposition Relations

Two-scale relations hold the basic relation in the MRA with the dyadic scaling factor. At each resolution level, the scaling function and the wavelets generate two bases by their discrete translates. The two-scale relations relate the scaling function and the wavelets at a given scale with the scaling function at the next-higher scale.

Let $\phi(t)$ be the basic scaling function that translates the subspace $\mathbf{V_0}$. The next finer resolution of subspace $\mathbf{V_1}$ is spanned by the set $\{\phi(2t-k)\}$, which is generated from the scaling function $\phi(t)$ by a contraction with a factor of two and by translation with half-integer steps. Note that the set $\{\phi(2t-k)\}$ can also be considered as a sum of two sets of

even and odd translates with integer steps, $\{\phi(2t-2k)\}$ and $\{\phi(2t-(2k+1))\}$. Since

$$\phi(t) \in \mathbf{V_0} \subset \mathbf{V_1} \tag{2.81}$$

$$\psi(t) \in \mathbf{W_0} \subset \mathbf{W_1} \tag{2.82}$$

It should be possible to write both functions $\phi(t)$ and $\psi(t)$ in terms of the bases that generate $\mathbf{V_1}$. This means that there exist two sequences $\{h[k]\}, \{g[k]\} \in \ell^2$ such that

$$\phi(t) = \sum_k h[k]\phi\left(2t-k\right) \tag{2.83}$$

$$\psi(t) = \sum_k g[k]\phi\left(2t-k\right) \tag{2.84}$$

Eq. (2.83) and **Eq**. (2.84) are referred to as *two-scale relations*. Discrete coefficient sequences $h[k]$ and $g[k]$ are called interscale coefficient that will be used in the wavelet decomposition as the discrete low-pass and high-pass filter respectively. Note that **Eq**. (2.84) is useful for generating wavelets from the scaling functions.

In general, for any $j \in \mathbb{Z}$, the relation between $\mathbf{V_j}$ and $\mathbf{W_j}$ with $\mathbf{V_{j+1}}$ is determined by

$$\phi\left(2^j t\right) = \sum_k h[k]\phi\left(2^{j+1}t-k\right) \tag{2.85}$$

$$\psi\left(2^j t\right) = \sum_k g[k]\phi\left(2^{j+1}t-k\right) \tag{2.86}$$

Similar to the two-scale relations, the decomposition relations define the scaling function at any given scale in terms of the scaling function and the wavelet at the next-lower scale. Because $\mathbf{V_1} = \mathbf{V_0} + \mathbf{W_0}$ and $\phi(2t), \phi(2t-1) \in \mathbf{V_1}$, there exist two sequences $\{\tilde{h}[k]\}, \{\tilde{g}[k]\} \in \ell^2$ such that

$$\phi(2t) = \sum_k \left\{\tilde{h}[2k]\phi(t-k) + \tilde{g}[2k]\psi(t-k)\right\} \tag{2.87}$$

$$\phi(2t-1) = \sum_k \left\{\tilde{h}[2k-1]\phi(t-k) + \tilde{g}[2k-1]\psi(t-k)\right\} \tag{2.88}$$

Combining these two relations leads to

$$\phi(2t-\ell) = \sum_k \left\{\tilde{h}[2k-\ell]\phi(t-k) + \tilde{g}[2k-\ell]\psi(t-k)\right\} \tag{2.89}$$

for all $\ell \in \mathbb{Z}$. Accordingly, the generalization of the decomposition relations is formulated as

$$\phi(2^{j+1}t-\ell) = \sum_k \left\{\tilde{h}[2k-\ell]\phi(2^j t-k) + \tilde{g}[2k-\ell]\psi(2^j t-k)\right\} \tag{2.90}$$

Famous but simple example to illustrate the two-scale relations and the decomposition relations is shown by taking the Haar wavelet as the case. The Haar scaling function is the simple rectangle function in the interval $[0, 1)$. It is defined as

$$\phi(t) = \begin{cases} 1 & \text{for } 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.91}$$

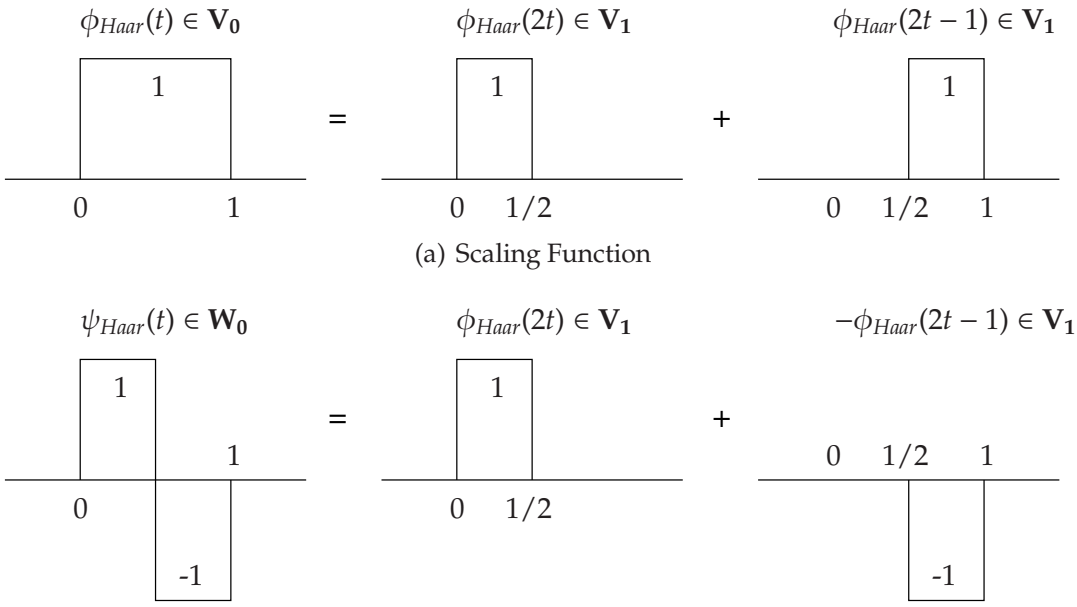


Fig. 2.6: Two-scale relations for Haar wavelet [?].

and its wavelet as

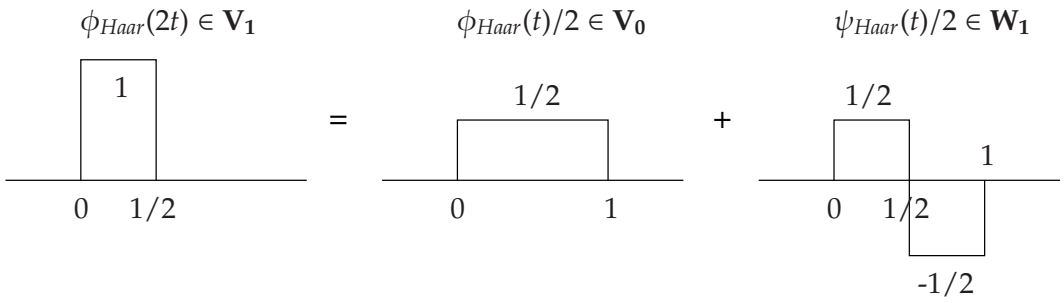
$$\psi(t) = \begin{cases} 1 & \text{for } 0 \le t < 1/2 \\ -1 & \text{for } 1/2 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.92}$$

Obviously, the integer step translations of this rectangular function generate an orthonormal set. The contracted Haar scaling function $\phi(2t)$ is a rectangular function in the interval $[0, 1/2)$. Its discrete translates with half integer steps generate an orthonormal basis $\{\phi(2t-k)\}$. As $\phi(2t-k)$ are half integer width, the $\phi(t) \in \mathbf{V_0}$ can be expressed as a linear combination of the even and odd translates of $\phi(2t) \in \mathbf{V_1}$. Two-scale relations for Haar wavelet are set by choosing $h[0] = h[1] = 1$, $g[0] = -g[1] = 1$, and $h[k] = g[k] = 0$ for all other k, which make

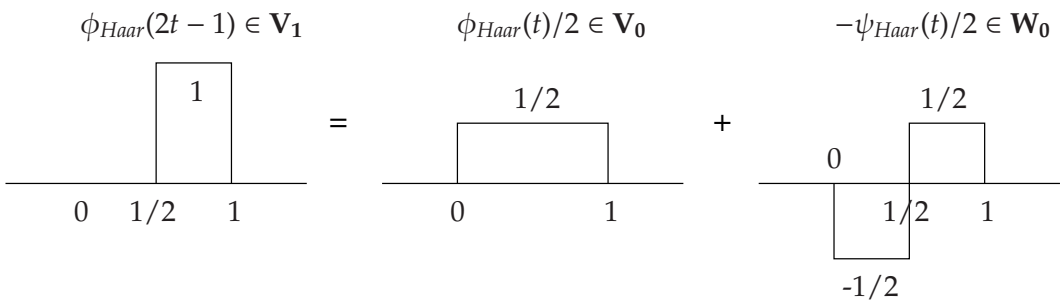
$$\phi(t) = \phi(2t) + \phi(2t-1) \tag{2.93}$$

$$\psi(t) = \phi(2t) - \phi(2t-1) \tag{2.94}$$

Fig. 2.6 illustrates the case. Likewise, decomposition relations for Haar wavelet are given by setting $\tilde{h}[0] = \tilde{h}[-1] = 1/2$, $\tilde{g}[0] = -\tilde{g}[1] = 1/2$, and $\tilde{h}[k] = \tilde{g}[k] = 0$ for all other k. **Fig**. 2.7 depicts the case for the decomposition of Haar scaling function.



(a) $\phi_{Haar}(2t)$



(b) $\phi_{Haar}(2t-1)$

Fig. 2.7: Decomposition relations for Haar wavelet [?].

2.8.1 Relation between Two-Scale Sequences

The Fourier transform of the two-scale relations defined in **Eq**. (2.83) and **Eq**. (2.84) are

$$\Phi(\omega) = H(z)\Phi\left(\frac{\omega}{2}\right) \tag{2.95}$$

$$\Psi(\omega) = G(z)\Phi\left(\frac{\omega}{2}\right) \tag{2.96}$$

with

$$H(z) = \frac{1}{2}\sum_k h[k]z^{k/2} \tag{2.97}$$

$$G(z) = \frac{1}{2}\sum_k g[k]z^{k/2} \tag{2.98}$$

where $z = e^{-j\omega}$. The orthogonality condition implies that for any $k \in \mathbb{Z}$,

$$\left\langle \phi(t-k), \psi(t) \right\rangle = 0 \tag{2.99}$$

By using the Parseval's identity, **Eq**. (2.99) can be written as

$$\begin{aligned} 0 &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \Phi(\omega)\Psi^*(\omega)e^{-j\omega k}\, d\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} H(z)G^*(z)\left|\Phi\left(\frac{\omega}{2}\right)\right|^2 e^{-j\omega k}\, d\omega \end{aligned}$$

$$= \frac{1}{2\pi} \sum_n \int_{4\pi n}^{4\pi(n+1)} H(z)G^*(z) \left|\Phi\left(\frac{\omega}{2}\right)\right|^2 e^{-j\omega k}\, d\omega$$
$$= \frac{1}{2\pi} \sum_n \int_0^{4\pi} H(z)G^*(z) \left|\Phi\left(\frac{\omega}{2} + 2\pi n\right)\right|^2 e^{-j\omega k}\, d\omega$$
$$= \frac{1}{2\pi} \int_0^{4\pi} H(z)G^*(z)E_\phi(z)e^{-j\omega k}\, d\omega \tag{2.100}$$

By partitioning the integration limit $[0, 4\pi]$ into $[0, 2\pi]$ and $[2\pi, 4\pi]$, and with a variable change, **Eq**. (2.100) becomes

$$\frac{1}{2\pi} \int_0^{2\pi} \left[H(z)G^*(z)E_\phi(z) + H(-z)G^*(-z)E_\phi(-z)\right] e^{-j\omega k}\, d\omega = 0 \tag{2.101}$$

From the discussion of the Fourier series in **App**. A, Fourier series representation of an integrable 2π-periodic function $f(t)$ is formulated as

$$f(\omega) = \sum_k c_k e^{j\omega k} \tag{2.102}$$

with

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(\omega)e^{-j\omega k}\, d\omega \tag{2.103}$$

Comparing the above equation with **Eq**. (2.101), the left part of **Eq**. (2.101) represents the k-th Fourier coefficient of a periodic function

$$H(z)G^*(z)E_\phi(z) + H(-z)G^*(-z)E_\phi(-z) \tag{2.104}$$

As all of these coefficients are zero, it implies that

$$H(z)G^*(z)E_\phi(z) + H(-z)G^*(-z)E_\phi(-z) \equiv 0 \tag{2.105}$$

for $|z| = 1$. **Eq**. (2.105) gives us the relation between $H(z)$ and $G(z)$. By direct substitution, **Eq**. (2.105) is simplified to

$$G(z) = -cz^{2m+1}H^*(-z)E_\phi(-z) \tag{2.106}$$

for a constant $c > 0$ and any integer m. For simplicity, c can be set to 1. The value m represents the shifting of the index of the sequence $\{g[k]\}$. It is usually chosen such that the index begins with 0.

2.8.2 Relation between Reconstruction and Decomposition Sequences

The Fourier transform of the decomposition relations defined in **Eq**. (2.89) is

$$\frac{1}{2}\Phi\left(\frac{\omega}{2}\right)e^{-j\omega k/2} = \sum_k \tilde{h}[2k - \ell]e^{-j\omega k}\Phi(\omega) + \sum_k \tilde{g}[2k - \ell]e^{-j\omega k}\Psi(\omega)$$
$$= \left(H(z)\sum_k \tilde{h}[2k - \ell]e^{-j\omega k} + G(z)\sum_k \tilde{g}[2k - \ell]e^{-j\omega k}\right)\Phi\left(\frac{\omega}{2}\right) \tag{2.107}$$

which can be reduced to

$$\left(\sum_k \tilde{h}[2k-\ell]e^{-j\omega(2k-\ell)/2}\right)H(z)+\left(\sum_k \tilde{g}[2k-\ell]e^{-j\omega(2k-\ell)/2}\right)G(z)\equiv\frac{1}{2} \tag{2.108}$$

for $\ell \in \mathbb{Z}$.

Combining the Fourier transform of both relations leads to

$$[\tilde{H}(z)+\tilde{H}(-z)]H(z)+[\tilde{G}(z)+\tilde{G}(-z)]G(z)=\frac{1}{2} \qquad \text{for even } \ell \tag{2.109}$$

$$[\tilde{H}(z)+\tilde{H}(-z)]H(z)+[\tilde{G}(z)+\tilde{G}(-z)]G(z)=0 \qquad \text{for odd } \ell \tag{2.110}$$

which gives us

$$H(z)\tilde{H}(z)+G(z)\tilde{G}(z)=\frac{1}{2} \tag{2.111}$$

$$H(z)\tilde{H}(-z)+G(z)\tilde{G}(-z)=0 \tag{2.112}$$

Eq. (2.112) can also be written as

$$H(-z)\tilde{H}(z)+G(-z)\tilde{G}(z)=0 \tag{2.113}$$

Rewriting **Eq**. (2.111) and **Eq**. (2.113) in matrix form,

$$\begin{bmatrix} H(z) & G(z) \\ H(-z) & G(-z) \end{bmatrix}\begin{bmatrix} \tilde{H}(z) \\ \tilde{G}(z) \end{bmatrix}=\begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix} \tag{2.114}$$

The solution for $\tilde{H}(z)$ and $\tilde{G}(z)$ are given by

$$\tilde{H}(z)=\frac{1}{2}\frac{G(-z)}{\Delta_{GH}(z)} \tag{2.115}$$

$$\tilde{G}(z)=-\frac{1}{2}\frac{H(-z)}{\Delta_{GH}(z)} \tag{2.116}$$

with

$$\Delta_{GH}(z)=H(z)G(-z)-H(-z)G(z) \tag{2.117}$$

In [?], it can be shown that

$$\Delta_{GH}(z)=cz^m E_\phi(z^2) \tag{2.118}$$

with $c>0$ and m is an integer number. Because ϕ generates a stable basis, $E_\phi(z)$ is guaranteed to be non-zero. This makes $\Delta_{GH}(z)\neq 0$.

2.9 Filter Banks

The ideas of signal expansions are linked to signal analysis, approximation, and compression, as well as their algorithms and implementations. Linear signal expansions used

in digital signal processing are mainly based on the block transforms [**?**, **?**, **?**, **?**]. Further researches in signal expansion, especially in orthogonal and biorthogonal bases, are initiated from the field of digital signal compression, where redundancy of the expansion has to be avoided. The *Quadrature Mirror Filters* (QMF) introduced by Croisier *et al.* in 1976 [**?**] was a key step forward, which allows a signal to be split into two downsampled subband signals and then reconstructed without introducing aliasing effect, even though non-ideal filters are used. The development leads to the founding of the perfect reconstruction filter banks. The first two-channel orthogonal solution was discovered by Smith and Barnwell [**?**,**?**] and Mintzer [**?**]. Viadyanathan expands the domain into multiple filter banks in [**?**, **?**, **?**]. Veterrli introduced biorthogonal solutions in [**?**] and also multidimensional quadrature mirror filters in [**?**]. Multidimensional filter banks were investigated in [**?**, **?**, **?**] and linear phase biorthogonal filters were further studied in [**?**, **?**]. Other breakthroughs in the area of multiresolution signal processing were contributed by Daubechies and Mallat. Daubechies showed that the wavelet can be reconstructed from filter banks in [**?**], whereas Mallat showed that a wavelet series expansion could be implemented with filter banks in [**?**]. From this brief discussion, two different points of perspective on the MRA exist, namely, expansion of signals in term of structured bases, and perfect reconstruction filter banks. The signal expansions deal more with Fourier and wavelet theory while the filter banks concentrate more into the construction of implementable systems.

Multiresolution signal processing deals with discrete-time sequences that are taken at different rates. It is always possible to go back to an underlying continuous-time signal and resample it at a different date. Nevertheless, most often the rate changes are performed in the discrete-time domain.

2.9.1 Decimation and Interpolation

In the field of multirate signal processing, the signal representation can have more than one sampling rate. The mechanisms for changing the sample rate are called decimation and interpolation.

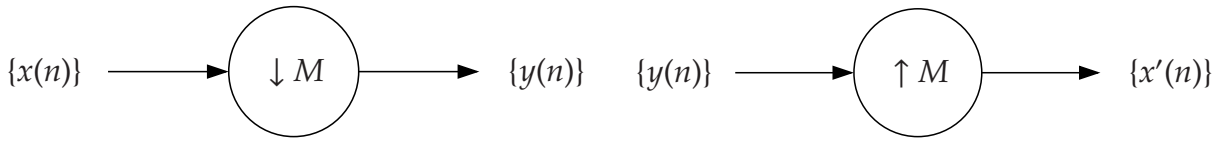
An M-point decimation of a function is formulated as

$$y(n) = x(nM) \qquad \text{for } n \in \mathbb{Z} \tag{2.119}$$

and its z-transform as

$$\begin{aligned} Y(z) &= \frac{1}{M}\sum_{k=0}^{M-1} X\left(z^{1/M}e^{-j2\pi k/M}\right) \\ &= \frac{1}{M}\sum_{k=0}^{M-1} X\left(z^{1/M}W_M^k\right) \end{aligned} \tag{2.120}$$

with $W_M^k = e^{-j2\pi k/M}$. **Fig**. 2.8(a) depicts the system diagram of an M-point decimator. For



(a) M-point decimator (b) M-point interpolator

Fig. 2.8: Diagram of decimator and interpolator [?].

octave scales, 2 is chosen as a decimator value for M, which makes

$$Y(z) = \frac{1}{2}\left[X\left(z^{1/2}\right) + X\left(-z^{1/2}\right)\right] \tag{2.121}$$

Decimator introduces aliasing effect if the spectral bandwidth is greater than π/M, i.e. $|\omega| > \pi/M$.

An M-point interpolator of a function is formulated as

$$x'(n) = \begin{cases} y\left(\frac{n}{M}\right) & \text{for } n = kM,\ k \in \mathbb{Z} \\ 0 & \text{elsewhere} \end{cases} \tag{2.122}$$

and its z-transform as

$$X'(z) = Y\left(z^M\right) \tag{2.123}$$

Fig. 2.8(b) depicts the system diagram of an M-point interpolator. Interpolator is a safe operator because the output spectrum has narrower bandwidth compared to the input spectrum, thus no danger of aliasing is introduced by the interpolator.

2.9.2 Decomposition and Reconstruction Algorithms

The decomposition algorithm is used to decompose the signal into MRA space. From the discussion of MRA, we have

$$x_{j+1}(t) \in \mathbf{V_{j+1}} \Longrightarrow x_{j+1}(t) = \sum_k v_{j+1,k}\phi_{j+1,k}(t) \tag{2.124}$$

$$x_j(t) \in \mathbf{V_j} \Longrightarrow x_j(t) = \sum_k v_{j,k}\phi_{j,k}(t) \tag{2.125}$$

$$y_j(t) \in \mathbf{W_j} \Longrightarrow y_j(t) = \sum_k w_{j,k}\psi_{j,k}(t) \tag{2.126}$$

and

$$x_{j+1}(t) = x_j(t) + y_j(t) \tag{2.127}$$

$$\sum_k v_{j+1,k}\phi_{j+1,k}(t) = \sum_k v_{j,k}\phi_{j,k}(t) + \sum_k w_{j,k}\psi_{j,k}(t) \tag{2.128}$$

Substituting the decomposition relation

$$\phi(2^{j+1}t - \ell) = \sum_k \left(\tilde{h}[2k - \ell]\phi(2^j t - k) + \tilde{g}[2k - \ell]\psi(2^j t - k)\right) \tag{2.129}$$

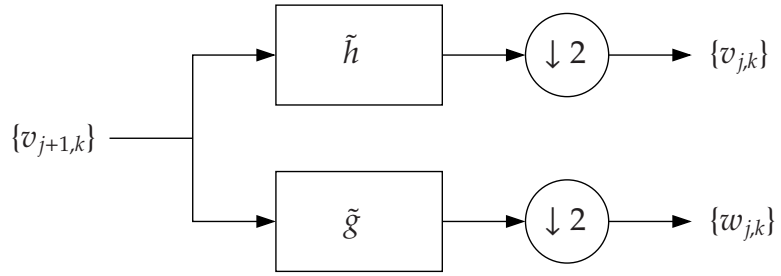


Fig. 2.9: Decomposition process for a one-level wavelet decomposition tree [?].

into **Eq**. (2.128), changing the order of summations, and comparing the coefficients of $\phi_{j,k}(t)$ and $\psi_{j,k}(t)$ leads to

$$v_{j,k} = \sum_{\ell} \tilde{h}[2k-\ell] v_{j+1,\ell} \tag{2.130}$$

$$w_{j,k} = \sum_{\ell} \tilde{g}[2k-\ell] v_{j+1,\ell} \tag{2.131}$$

Note that the right side of the equations correspond to decimation by two after convolution. These equations can be illustrated in **Fig**. 2.9. This decomposition structure will be nested on the $\{v_{j,k}\}$ in order to achieve the wavelet decomposition tree. It is obvious that the generated decomposition tree will be non symmetric in this case. A symmetric decomposition tree can be achieved by nesting also the decomposition on the $\{w_{j,k}\}$. This leads to the wavelet packet decomposition tree.

Reconstruction algorithm is used to reconstruct the decomposed signal back perfectly. Similar to the decomposition algorithm, the reconstruction algorithm is based on two-scale relations of the scaling function and the wavelet. Substituting the two-scale relations into **Eq**. (2.128) leads to

$$\sum_{k} v_{j,k} \sum_{\ell} h[\ell]\phi(2^{j+1}t-2k-\ell) + \sum_{k} w_{j,k} \sum_{\ell} g[\ell]\phi(2^{j+1}t-2k-\ell) = \sum_{\ell} v_{j+1,k}\phi(2^{j+1}t-\ell) \tag{2.132}$$

By comparing the coefficients of $\phi(2^{j+1}t-\ell)$ on both sides, we have

$$v_{j+1,\ell} = \sum_{k} \Big(h[\ell-2k] v_{j,k} + g[\ell-2k] \Big) \tag{2.133}$$

which corresponds to interpolation by two, followed by convolution. **Fig**. 2.10 depicts the illustration of the reconstruction algorithm.

From the discussion, it is clear that digital filter can be applied in order to decompose the signal into wavelet space and to recover the signal perfectly from the wavelet space. Although the concept of the filter banks is general, the discussion is limited to the two-channel filter banks as in case for discrete wavelet transform with octave scales.

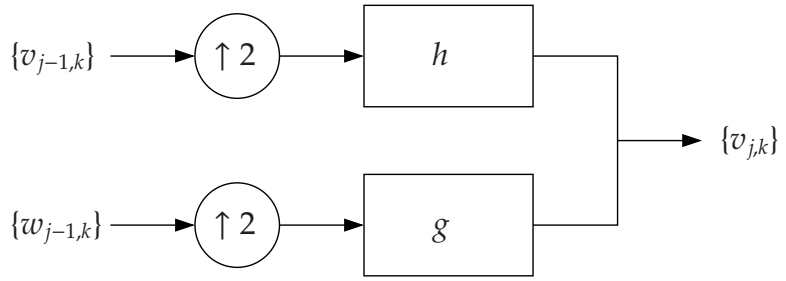


Fig. 2.10: Reconstruction process for a one-level wavelet reconstruction tree [?].

2.9.3 Two-Channel Perfect Reconstruction Filter Bank

Filter bank is a collection of multiple bandpass filter used to separate a signal into different components whose spectra occupy different segments of the frequency axes. Some applications that use filter bank are Doppler frequencies in radar signal processing [**?**, **?**] and equalizer in music signal processing [**?**, **?**]. Perfect reconstruction filter banks, as the name says, are filter banks that are specially designed so that the signal can be reconstructed perfectly without any information loss.

Let $X(z)$ be the input of the two-channel filter banks, $\tilde{H}(z)$ and $\tilde{G}(z)$ be the analysis filter pairs. $V(z)$, the output of the signal after low-pass filter $\tilde{H}(z)$ and decimation by two, is written as

$$V(z) = \frac{1}{2}\left[X(z^{1/2})\tilde{H}(z^{1/2}) + X(-z^{1/2})\tilde{H}(-z^{1/2})\right] \tag{2.134}$$

$W(z)$, the output of the signal after high-pass filter $\tilde{G}(z)$ and decimation by two, is written as

$$W(z) = \frac{1}{2}\left[X(z^{1/2})\tilde{G}(z^{1/2}) + X(-z^{1/2})\tilde{G}(-z^{1/2})\right] \tag{2.135}$$

Let $H(z)$ and $G(z)$ be the synthesis filter pairs, $V'(z)$ be the output of $V(z)$ after interpolation by two and $H(z)$, and $W'(z)$ be the output of $W(z)$ after interpolation by two and $G(z)$.

$$V'(z) = \frac{1}{2}\left[X(z)\tilde{H}(z)H(z) + X(-z)\tilde{H}(-z)H(z)\right] \tag{2.136}$$

$$W'(z) = \frac{1}{2}\left[X(z)\tilde{G}(z)G(z) + X(-z)\tilde{G}(-z)G(z)\right] \tag{2.137}$$

$X'(z)$, the combined output of $V'(z)$ and $W'(z)$ is written as

$$\begin{aligned} X'(z) &= V'(z) + W'(z) \\ &= \frac{1}{2}X(z)\left[\tilde{H}(z)H(z) + \tilde{G}(z)G(z)\right] + \frac{1}{2}X(-z)\left[\tilde{H}(-z)H(z) + \tilde{G}(-z)G(z)\right] \end{aligned} \tag{2.138}$$

The last term contributes to the aliasing. Fig 2.11 depicts the two-channel filter bank that consists of analysis and synthesis filter pairs.

In order to eliminate the aliased version of the reconstructed signal, the last term must

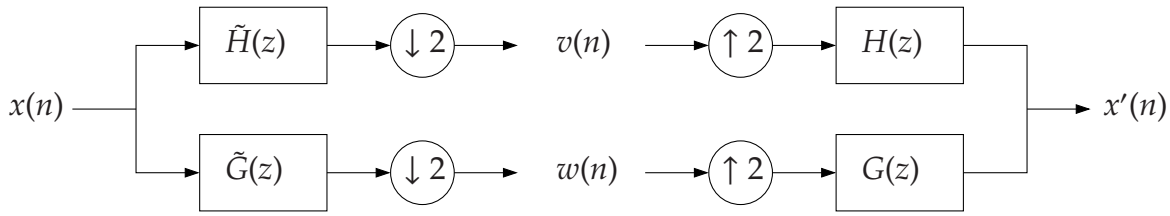


Fig. 2.11: Two-channel filter bank [?].

be cancelled. It can be achieved by choosing

$$H(z) = \pm\tilde{G}(-z) \tag{2.139}$$

$$G(z) = \mp\tilde{H}(-z) \tag{2.140}$$

Thus, $X'(z)$ becomes

$$X'(z) = \frac{1}{2}X(z)\left[\tilde{H}(z)\tilde{G}(-z) - \tilde{H}(-z)\tilde{G}(z)\right] \tag{2.141}$$

Perfect reconstruction means that the reconstructed signal $X'(z)$ can only be a delayed version of the input signal $X(z)$, i.e. its transfer function $T(z)$ must satisfy

$$T(z) = \frac{X'(z)}{X(z)} = z^{-m} \qquad m \in \mathbb{Z}^+ \tag{2.142}$$

which makes

$$2z^{-m} = \tilde{H}(z)\tilde{G}(-z) - \tilde{H}(-z)\tilde{G}(z) \tag{2.143}$$

$$= \tilde{H}(z)H(z) - \tilde{H}(-z)H(-z) \tag{2.144}$$

$$= \tilde{H}(z)H(z) + \tilde{G}(z)G(z) \tag{2.145}$$

For the sake of simplicity, we can define composite filters $C_0(z)$ and $C_1(z)$ as product filters for the two filtering paths

$$C_0(z) = \tilde{H}(z)H(z) = -\tilde{H}(z)\tilde{G}(-z) \tag{2.146}$$

$$C_1(z) = \tilde{G}(z)G(z) = \tilde{H}(z)\tilde{G}(z) = -C_0(-z) \tag{2.147}$$

The perfect reconstruction condition becomes

$$C_0(z) - C_0(-z) = 2z^{-m} \tag{2.148}$$

and

$$T(z) = \frac{1}{2}\left[C_0(z) - C_0(-z)\right] \tag{2.149}$$

Since

$$T(-z) = \frac{1}{2}\left[C_0(-z) - C_0(z)\right] = -T(z) \tag{2.150}$$

it states that the transfer function $T(z)$ is an odd function. Thus, the integer m must be chosen to be odd. It implies that $C_0(z)$ must contain only even-indexed coefficients except for $c_m = 1$. Finding $\tilde{H}(z)$ and $\tilde{G}(z)$ to satisfy the perfect reconstruction condition is the subject of filter bank design and it is out of the scope of this thesis. The readers are suggested to read the textbooks [?, ?, **?**, ?, ?, ?].

2.10 Polyphase Representation for Filter Banks

Performing discrete wavelet transforms using filter banks require convolution with low-pass and high-pass analysis filter pair, followed by decimation on both outputs with a factor of two. The reconstruction of the signal is performed inversely. It starts with interpolation with a factor of two, followed by low-pass and high-pass synthesis filter pair. The implementation using filter banks is straightforward. Note that decimation by two means that the output of the decimator retains only every second sample of the input signal. While performing the wavelet decomposition, it is apparent that half of the output of the analysis filters fed to the decimator is thrown away. This make the implementation of wavelet transform using filter banks not optimal.

Polyphase representation of a signal is an alternative approach to represent a signal other than time and spectral domains. In the polyphase approach, the decimation by two comes in advance, followed by convolution. Thus, the convolution is performed only with half of the coefficients. This approach increases the efficiency of the computation by reducing the data redundancy.

Consider the upper path with input $X(z)$ and filter $\tilde{H}(z)$ from **Fig**. 2.11. The output of the filtering needs to be downsampled by two, i.e. we want to keep the *even powers* of z in the product $\tilde{H}(z)X(z)$. However, the even powers come from even powers in $\tilde{H}(z)$ times even power in $X(z)$ and from odd powers in $\tilde{H}(z)$ times odd powers in $X(z)$.

The separation of even and odd powers is written as

$$X(z) = X_e(z^2) + z^{-1}X_o(z^2) \tag{2.151}$$

$$\tilde{H}(z) = \tilde{H}_e(z^2) + z^{-1}\tilde{H}_o(z^2) \tag{2.152}$$

where the subscript e and o stand for *even* and *odd* respectively. The even powers of the product $\tilde{H}(z)X(z)$ are written as

$$\left(\tilde{H}(z)X(z)\right)_e(z^2) = \tilde{H}_e(z^2)X_e(z^2) + z^{-2}\tilde{H}_o(z^2)X_o(z^2) \tag{2.153}$$

The downsampling by two changes z^2 to z, thus

$$V(z) = (\downarrow 2)\tilde{H}(z)X(z) = \tilde{H}_e(z)X_e(z) + z^{-1}\tilde{H}_o(z)X_o(z) \tag{2.154}$$

Here, in order to compute $v(n)$, instead of filtering and followed by downsampling, first we separate the input sequence $x(n)$ into even samples $x_e(n)$ and odd samples $x_0(n)$, followed by downsampling and the filtering. Similarly, this principle can be applied on $\tilde{G}(z)$. Using matrix notation, we have

$$\begin{bmatrix} V(z) \\ W(z) \end{bmatrix} = \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}_e(z) & \tilde{G}_o(z) \end{bmatrix} \begin{bmatrix} X(z) \\ z^{-1}X(z) \end{bmatrix}$$

$$= \mathbf{P}(z) \begin{bmatrix} X(z) \\ z^{-1}X(z) \end{bmatrix} \tag{2.155}$$

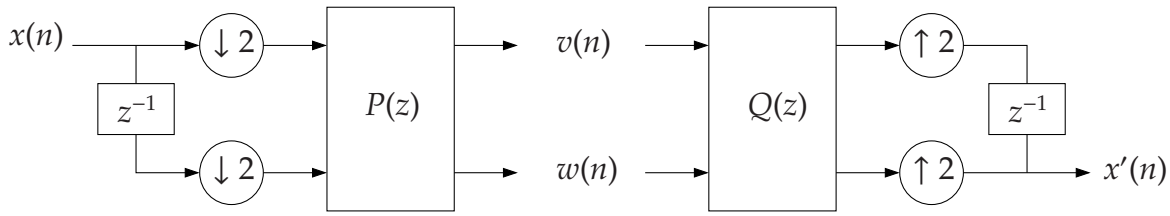


Fig. 2.12: Polyphase realization of a two-channel filter bank [?].

The same idea can be extended to synthesis the signal

$$\begin{aligned}\mathbf{X}'(z) &= \begin{bmatrix}1 & z^{-1}\end{bmatrix}\begin{bmatrix}H_e(z^2) & G_e(z^2)\\ H_o(z^2) & G_o(z^2)\end{bmatrix}\begin{bmatrix}V(z^2)\\ W(z^2)\end{bmatrix}\\ &= \mathbf{Q}(z)\begin{bmatrix}V(z^2)\\ W(z^2)\end{bmatrix}\end{aligned} \tag{2.156}$$

Fig. 2.12 depicts the realization of a two-channel filter bank using polyphase approach.

To satisfy the perfect reconstruction condition, the polyphase matrices must satisfy

$$\mathbf{Q}(z)\mathbf{P}(z) = \mathbf{I} \tag{2.157}$$

In case of an orthogonal transform, the synthesis polyphase matrix is the transpose of the analysis polyphase matrix, that is

$$\mathbf{Q}(z) = \mathbf{P}(z^{-1})^T \tag{2.158}$$

In more general cases, the analysis polyphase matrix must be invertible. These filter banks are referred to as biorthogonal, written as

$$\mathbf{Q}(z) = \mathbf{P}(z)^{-1} \tag{2.159}$$

Let $N_{\tilde{H}}$ and $N_{\tilde{G}}$ be the length of the low-pass and high-pass analysis filter pair $\tilde{H}(z)$ and $\tilde{G}(z)$. The direct wavelet decomposition using filter bank approach requires $N_{\tilde{H}} + N_{\tilde{G}}$ multiplications per unit time. Because the downsampling takes place before the polyphase filter, wavelet decomposition using polyphase requires only half of the cost, i.e. $\frac{N_{\tilde{H}}+N_{\tilde{G}}}{2}$.

2.11 Lifting Scheme

Lifting scheme is a solution to implement two-channel filter bank in an efficient manner. The idea of the lifting scheme comes from the polyphase representation of the analysis and synthesis filter pair. Contrary to the filter approach, which separates the signal into low and high frequency components and performs the decimation on both signals afterwards, the second generation of wavelets is similar to the polyphase. It reduces the computation by performing the decimation in advance. The second generation of wavelets,

more popular under the name of lifting scheme, was introduced by Sweldens [**?**, ?]. The first generation or classical wavelets rely heavily on Fourier analysis whereas the second generation exploits wavelets adapted to situations that do not allow translation and dilation of the wavelets function. Thus, it allows to explain the basic ideas behind wavelets in a much simpler manner and also to define a wavelet basic on an interval or an irregular grid, or even on a sphere.

The concept of wavelet transform is to exploit the correlation present in signals to construct a sparse approximation. The wavelet transform on a one-dimensional signal is a multiresolution representation of the signal. The classical wavelets, as mentioned earlier, are coupled with the Fourier analysis. Here, decorrelation happens in frequency domain where at each decomposition level, the low-pass component of the signal is split into a low-pass and high-pass components. However, the decorrelation using lifting scheme is entirely spatial. Thus, it is perfectly suited for building second generation wavelets when Fourier techniques are no longer available.

Let $\mathbf{x} = \{x(n)\}, n \in \mathbb{Z}$ be a set of signal, $\mathbf{x_e} = \{x(2n)\}$ and $\mathbf{x_o} = \{x(2n+1)\}$ be its even and odd subset. This kind of splitting is called *lazy wavelet transform*. Given one set, e.g. even indexed samples, a good predictor $\mathscr{P}$ can be built to predict the other set. Because the predictor cannot be exact, we obtained the difference or detail $\mathbf{d}$, i.e.

$$\mathbf{d} = \mathbf{x_o} - \mathscr{P}(\mathbf{x_e}) \tag{2.160}$$

Eq. (2.160) is called *dual lifting step*. If the signal is highly correlated, we can expect the difference to be small, in other word, $\mathbf{d}$ is a sparse set. This transform maps $(\mathbf{x_e}, \mathbf{x_o})$ to $(\mathbf{x_e}, \mathbf{d})$. The new representation loses some desired properties such as the mean value of the original samples, which is no longer equal to the mean value of $\mathbf{x_e}$ due to downsampling. The *primal lifting* step is introduced to restore this property. It replaces $\mathbf{x_e}$ with the updated samples computed from $\mathbf{d}$ with update operator $\mathscr{U}$, formulated as

$$\mathbf{s} = \mathbf{x_e} + \mathscr{U}(\mathbf{d}) \tag{2.161}$$

We have now the transform from the pair $(\mathbf{x_e}, \mathbf{x_o})$ to $(\mathbf{s}, \mathbf{d})$. **Fig**. 2.13 depicts the block diagram of the wavelet decomposition using lifting scheme. Recall the discussion of MRA. Here, $x_j(n)$ represents the original signal, $x_{j-1}(n)$ and $y_{j-1}(n)$ represent the average $\mathbf{s}$ and the difference $\mathbf{d}$, which also correspond to the one level lower of the MRA space.

Both dual and primal lifting steps are invertible, i.e.

$$\mathbf{x_e} = \mathbf{s} - \mathscr{U}(\mathbf{d}) \tag{2.162}$$

$$\mathbf{x_o} = \mathscr{P}(\mathbf{x_e}) + \mathbf{d} \tag{2.163}$$

thus, it leads to perfect reconstruction filter banks. Since the construction did not use the Fourier transform, the resulting wavelet transform can be applied to irregularly spaced samples or used in higher dimensions and leads to wavelets on a sphere.

The basic principle of lifting scheme is to factorize the wavelet filter into alternating upper and lower triangular 2×2 matrix. Daubechies and Sweldens in [?, ?] have shown

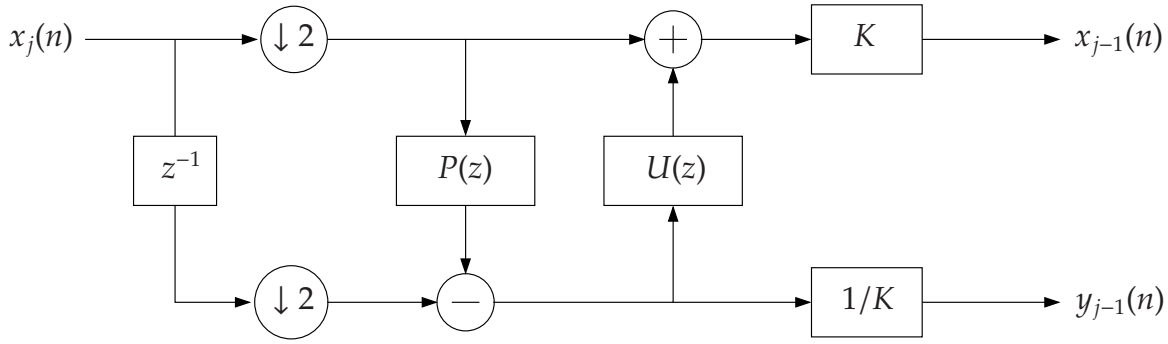


Fig. 2.13: Basic lifting scheme with prediction and update step [?].

that the polyphase representation can always be factored into lifting steps by using the Euclidean algorithm to find the greatest common divisors. Thus the polyphase representation becomes:

$$\mathbf{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{M} \begin{bmatrix} 1 & u_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ p_i(z) & 1 \end{bmatrix} \tag{2.164}$$

where $p_i(z)$ and $u_i(z)$ are the Laurent polynomials and K is the normalization factor. Given a FIR filter $H(z)$, its polynomial form is written as

$$h(z) = \sum_{k=k_\ell}^{k_r} h[k] z^{-k} \tag{2.165}$$

where k_ℓ and k_r are integers and satisfy $k_\ell \leq k_r$. Assuming that $h[k_\ell] \neq 0$ and $h[k_r] \neq 0$, the degree of the Laurent polynomial is given by

$$|h| = k_r - k_\ell \tag{2.166}$$

The zero Laurent polynomial degree is assigned to the degree $-\infty$ and a polynomial of degree zero is referred to as a monomial.

Fig. 2.14 shows the arrangement of the lifting scheme representation that is divided into several lifting steps. The Laurent polynomials $p_i(z)$ and $u_i(z)$ are expressed as predictor $P_i(z)$ and updater $U_i(z)$. The signal $x_j(n)$ is split into even and odd parts. Prediction and update steps occur alternately. The predictor $P_i(z)$ predicts the odd part from the even part. The difference between the odd part and the predicted part is computed and used by the updater $U_i(z)$ to update the even part. At the end, the low-pass and the high-pass signals are normalized with a factor of K and $1/K$ respectively.

By factoring the wavelet filters into lifting steps, it is expected that the computation performed on each stage (either it is a prediction or an update) will be much less complex. As an example, the famous Daub-4 wavelet filter with the low-pass filter response:

$$H(z) = \frac{1+\sqrt{3}}{4\sqrt{2}} + \frac{3+\sqrt{3}}{4\sqrt{2}} z^{-1} + \frac{3-\sqrt{3}}{4\sqrt{2}} z^{-2} + \frac{1-\sqrt{3}}{4\sqrt{2}} z^{-3} \tag{2.167}$$

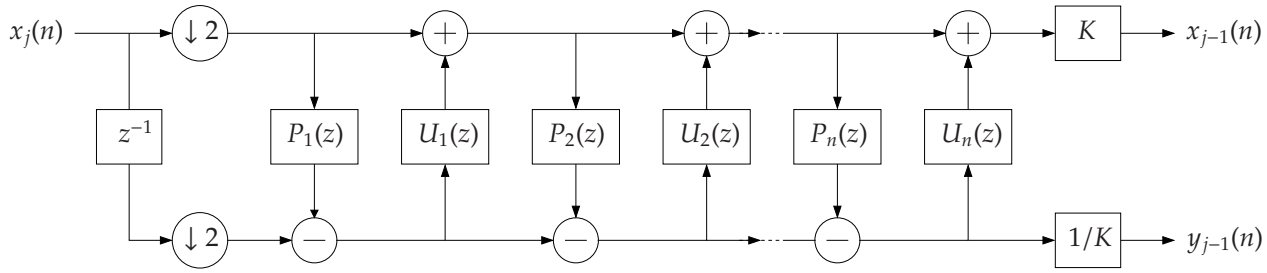


Fig. 2.14: Wavelet decomposition using lifting steps [?].

can be factored into lifting steps:

$$\mathbf{P}(z) = \begin{bmatrix} \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ 0 & \frac{\sqrt{3}+1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4} z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix} \tag{2.168}$$

App. C details the algorithm to factorize a wavelet filter into a lifting scheme. Since the finding of the greatest common divisors is not necessarily unique, the result of the factorization may also differ. The Daub-6 and the popular (5,3) and (9,7) wavelet filters can be factored into lifting steps with maximum degree of ±1 [?] whereas Symlet-6 and Coiflet-2 may have two update/prediction terms and also higher order z degree on its Laurent polynomials.

Chapter 3

State-of-the-Art in Discrete Wavelet Processors

Contents

3.1 Filter Banks Systolic-based Architectures 43

3.1.1 DWT-SA Architecture 43

3.1.2 Arc_J Architecture 46

3.2 Lifting-Based Architectures 48

3.2.1 Recursive Architecture 49

3.2.2 1D Folded Architecture 51

3.2.3 Row and Column Processors Architectures 53

3.3 Architectures for Computing Discrete Wavelet Packet 55

3.4 Concluding Remarks 57

The breakthrough in the wavelet theory which has been studied in the last two decades by many researchers [?,?,?,?,?] has delivered a solid background and has drawn much attention due to its attractive properties especially regarding its time-frequency localization feature [?, **?**]. The main study of the wavelet theory involves finding the answer for describing better and more appropriate functions to represent signals than the ones offered by the Fourier analysis.

Contrary to the Fourier analysis, which decomposes signals into sine and cosine functions, wavelets study each component of the signal on different resolutions and scales. In analogy, if we observe the signal with a large *window*, we will get a coarse feature of the signal, and if we observe the signal with a small *window*, we will extract the details of the signal. Whereas wavelets are based on the MRA which divides the signal into different scales of resolution rather than different frequencies, the original wavelet computation involves the convolution to filter the signal into different octave frequency bands. The

second generation of wavelets which was studied by Sweldens [?] offered less computation complexity by means of lifting scheme.

One of the most attractive features that wavelet transforms provide is their capability to analyze the signals which contain sharp spikes and discontinuities. Along with the better energy compacting feature, the wavelet transforms also support the localizing feature in both time and frequency domains [?]. These make wavelet outperforms the Fourier transform in signal processing and has made itself into the new standard of JPEG2000 [?, ?].

Along with recent trends and research focuses in applying wavelets in image processing, the application of wavelets is essentially not only limited to this area. The benefits of wavelets have been studied by many scientists from different fields such as mathematics, physics, and electrical engineering. In the field of electrical engineering wavelets have been known with the name multirate signal processing. Due to numerous interchanging fields, wavelets have been used in many applications such as image compression, feature detection, seismic geology, human vision, etc. Contrary to the Fourier transform, which uses one basis function (and its inverse) to transform between domains, there are different classes of wavelet kernels which can be applied on the signal depending on the application. Because different applications require different treatments, researchers have tried to cope with their own issues and implemented only a subset of wavelets which are suitable for their own needs such as ones that can be found in image compression [?,?,?,?], speech processing [?,?,?,?], feature extraction and detection [**?**], biomedicine [**?**], and statistics [**?**]. The power of wavelet tools is then limited due to these approaches.

In recent years, wavelets have become a hot topic in both industry and research fields due to the adoption of the wavelet transforms by the JPEG committee in their latest JPEG2000 standard [**?**, ?, ?, **?**, ?, **?**, ?, **?**, ?]. Since the introduction of the JPEG standard in 1992, the algorithms to compute Fast Fourier Transform (FFT) that is used on its transform block are extensively studied. Optimization on butterfly operations, different types of radix, addressing mechanisms, and register allocations are some of the main research focuses on designing algorithms that compute FFT in hardware in an efficient manner. Contrary to the FFT block in JPEG that tiles the image into 8×8 matrix, DWT in JPEG2000 uses much larger matrix, i.e. 256×256 in order to reduce the aliasing effect when the image is compressed with higher compression rates. Efficient hardware realizations of the wavelet transforms become an important issue because wavelet transforms deal with larger number of samples.

Although wavelet transforms are heavily used in JPEG2000, the applications of the wavelet transforms are not only limited to image compression. Nonetheless, there exist quite few hardware architectures that deal with other applications besides JPEG2000. Depending on the application's need, wavelet transforms have several operational modes. In the traditional wavelet transforms, wavelet decomposition, that is forward DWT, decomposes the signal into two components, i.e. low-pass and high-pass components. The resulting low-pass component is used to generate MRA space for the next level and so on.

The wavelet reconstruction, that is inverse DWT, reconstructs the signal from the highest available level by using both low-pass and high-pass components of the same level in order to reconstruct MRA space for the lower level and so on. This traditional wavelet transforms generate octave-spaced frequency bands. If we want to have an equally-spaced frequency band using the wavelet transforms, the resulting high-pass component needs also to be decomposed on each level. As an example, 3-level wavelet decomposition using traditional scheme generates four frequency bands whilst 3-level wavelet decomposition using the later technique generates eight equally-spaced frequency bands. This scheme is called wavelet packet decomposition and reconstruction, or in the discrete-time domain, it is called Discrete Wavelet Packet (DWP).

In this chapter, we discuss several existing hardware architectures for both DWT and DWP. As wavelet transforms are heavily used in JPEG2000 in recent years, we also focus our discussion on these existing architectures. **Sec**. 3.1 details several architectures based on filter banks and **Sec**. 3.2 details the lifting-based architectures to compute DWT decomposition and reconstruction. Finally, **Sec**. 3.3 closes the discussion with the DWP architectures.

3.1 Filter Banks Systolic-based Architectures

The architecture of the hardware realization of DWT went back as far as 90s. Several architectural solutions using special purpose parallel processors have been developed [**?**, ?, **?**,**?**,**?**,**?**,**?**] in order to meet the real-time requirement in many applications. These solutions include parallel filter architectures, SIMD (Single Instruction Multiple Data) linear array architectures, SIMD multigrid architectures, 2D block based architectures, etc. and they are in general based on the filter banks.

Before we discuss the more advanced architectures which are based on lifting scheme initiated by Sweldens, we go back to the VLSI realizations of wavelet transform using filter banks. The advantages of the filter-based implementation are that the architecture is straightforward and changing the wavelet kernel does not alter the architecture so much, because on most architectures, we only need to change the wavelet coefficients and to adapt the length of the wavelet filter during the design time.

3.1.1 DWT-SA Architecture

Grzeszczak *et al.* [?] present a design and VLSI implementation of an efficient systolic array architecture for computing DWT, called DWT-SA. The architecture computes both low-pass and high-pass frequency coefficients in the same clock cycle and thus has an efficient hardware utilization. The design is simple, modular and cascadable for computing of 1D and 2D data streams of fairly arbitrary size. The architecture is an improvement over the digit serial wavelet architecture using the pyramid algorithm outlined in [?]. The

improvement is achieved by using only one set of multipliers and adders instead of employing two parallel computational hardware [?]. The DWT-SA architecture does not use any external or internal memory modules to store the intermediate results and therefore avoids the delays caused by memory access and memory refresh timing. In addition, since a set of registers controlled by a global clock is employed, the control circuitry does not need to take the intermediate products in and out of the memory. This results in a simple and efficient systolic implementation for 1D DWT computation.

The design of DWT-SA is based on computation schedules derived for Daub-3 wavelet filter with 6 tap filter, which are the results of applying the pyramid algorithm for eight data points. Therefore, the architecture is a complete setup for computing DWT decomposition. The DWT reconstruction can be calculated by replacing the decimator with an interpolator. **Fig**. 3.1 depicts the block diagram of DWT-SA architecture. It comprises of four basic units, i.e. input delay, filter, register bank, and control unit. The filter unit is a six tap nonrecursive FIR digital filter with multiplexed low-pass and high-pass coefficients, as shown in **Fig**. 3.2. Computation of any DWT coefficient can be executed by employing a multiply and accumulate operation where partial products are computed separately and subsequently added. This feature makes possible systolic implementation of a DWT. The latency of each filter stage is one clock cycle.

By introducing additional control circuitry, computations of both low-pass and high-pass DWT coefficients can be executed using the same hardware in one clock cycle. The high-pass coefficient calculation is performed during the first half of the clock cycle, whereas the low-pass coefficient calculation is performed during the second half. Subsequently, the partial results are passed synchronously in a systolic manner from one cell to the adjacent cell. Therefore the architecture consists of only one multiplier, one adder, and two registers to store the low-pass and high-pass coefficients on each filter unit.

Two storage units are used in this DWT-SA, namely input delay and register bank. Based on the filter approach, the input delay unit stores the data samples that comprise of five previous samples for Daub-3 filter. Therefore, five data registers are connected serially in a chain. Register bank unit has a size of 26 which is used as storage of the intermediate partial results.

The control unit controls the multilevel decomposition. Assuming that N coefficients need to be processed, the first octave computations are scheduled every $N/4$, while the second and the third octave are schedule every $N/2$ and N clock cycles respectively. The details of the scheduling can be found in [?]. The main idea is to provide schedule allocation of the register bank so that the interconnect with the multiplexers that control the multilevel decomposition can be derived. Therefore, even the architecture is modular and cascadable, it can be designed only for one specific wavelet filter. If other filters are used, the interconnect between multiplexers and the register banks need to be redesigned. Additionally, this architecture can only perform DWT decomposition and the depth of the multilevel decomposition is restricted into 3 levels. The register bank grows exponentially in order to make the architecture compute higher decomposition level. Also, the multi-

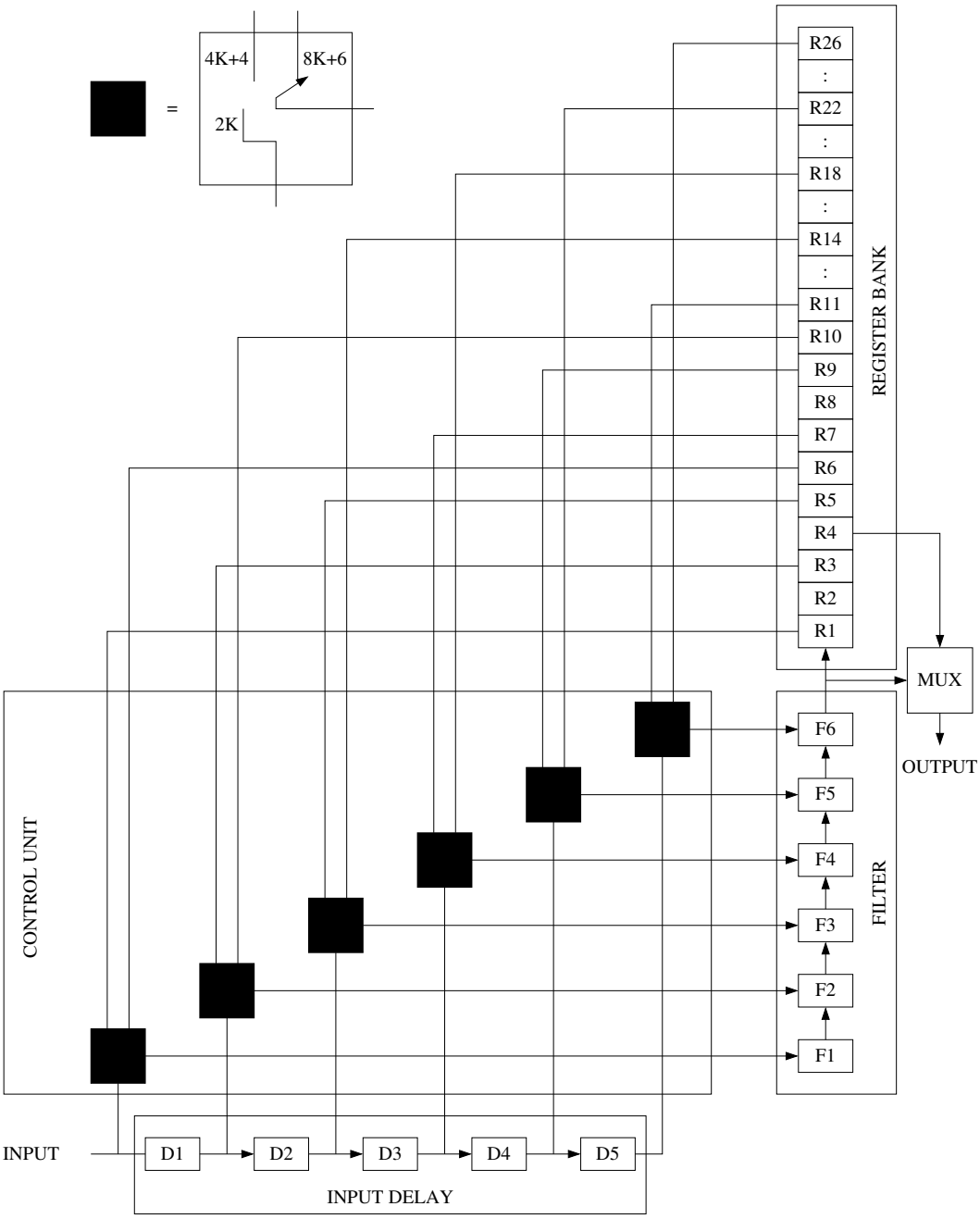


Fig. 3.1: DWT-SA architecture proposed by Grzeszczak [?].

plexer inputs need to be adapted to cope with this issue and additional interconnects need to be established. Last but not least, DWT-SA architecture does not include either memory or memory interface. It assumes that the samples are streamed to its input port at certain time unit. Therefore, in order to use this architecture to compute N-dimensional DWT decomposition, external interface that talks with the memory needs to be created. As we can see later in **Chap. 4**, the design of the memory controller cannot be neglected, because even the idea of providing samples is straightforward, its implementation is not. It is due to the fact that computing the wavelet coefficients in the boundary regions needs

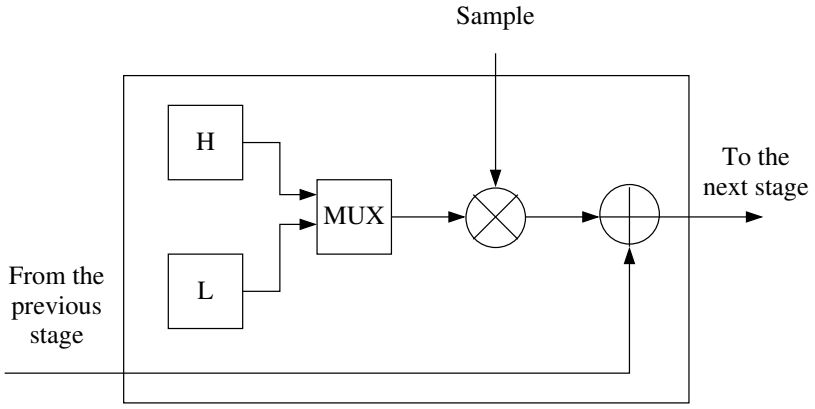


Fig. 3.2: Filter unit with low-pass and high-pass coefficients selector [?].

special treatments in order to make the decomposition invertible without growing the length of the resulting wavelet coefficients.

3.1.2 Arc$_J$ Architecture

Another systolic-based DWT implementation is developed by Marino *et al.* [?]. The architecture is called Arc$_J$. Actually, they design two types of architectures for computing 1D DWT, namely Arc$_J$ and Arc$_2^*$. The first architecture handles with high-speed data demands while the later tries to target the low-power applications by reducing the operating speed by half.

Because of its structure, 1D DWT can be straightforwardly pipelined into J blocks B_j with $1 \leq j \leq J$, each block B_j being devoted to compute the decomposition level j. From the DWT decomposition algorithm, we can clearly see that the complexity of the decomposition level j is linear in N_j with a factor depending on the sample length. Therefore, because of the decimation by two performed in each decomposition level, the complexity of the next decomposition level is half of the complexity of the current level. As a consequence, in order to balance a pipelined DWT architecture, each block B_j should employ a number W_j of processing elements (PEs), with $W_j = W_{j-1}/2$. Therefore, the idea of this architecture is to build a balanced Arc$_J$ constituted by a pipeline $\{B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_J\}$. Each block B_j is designed carefully by taking into account this condition. A top-level scheme of Arc$_J$ is given in **Fig**. 3.3. This block is responsible for computing the first DWT decomposition stage.

In order to design a high-speed block B_1 performing the first level of DWT decomposition, the architecture is split into four independent filter units, i.e. $\mathbf{A}^{EVEN}$, $\mathbf{A}^{ODD}$, $\mathbf{C}^{EVEN}$, and $\mathbf{C}^{ODD}$, which correspond to low-pass and high-pass even and odd components. These filter units employ $2L$ PEs globally in order to perform the first level of DWT at the rate of two samples per clock cycle, with the assumption that one clock cycle is the time required by one PE to compute one product and one sum. Block B_1 is shown in **Fig**. 3.3 for $L = 6$.

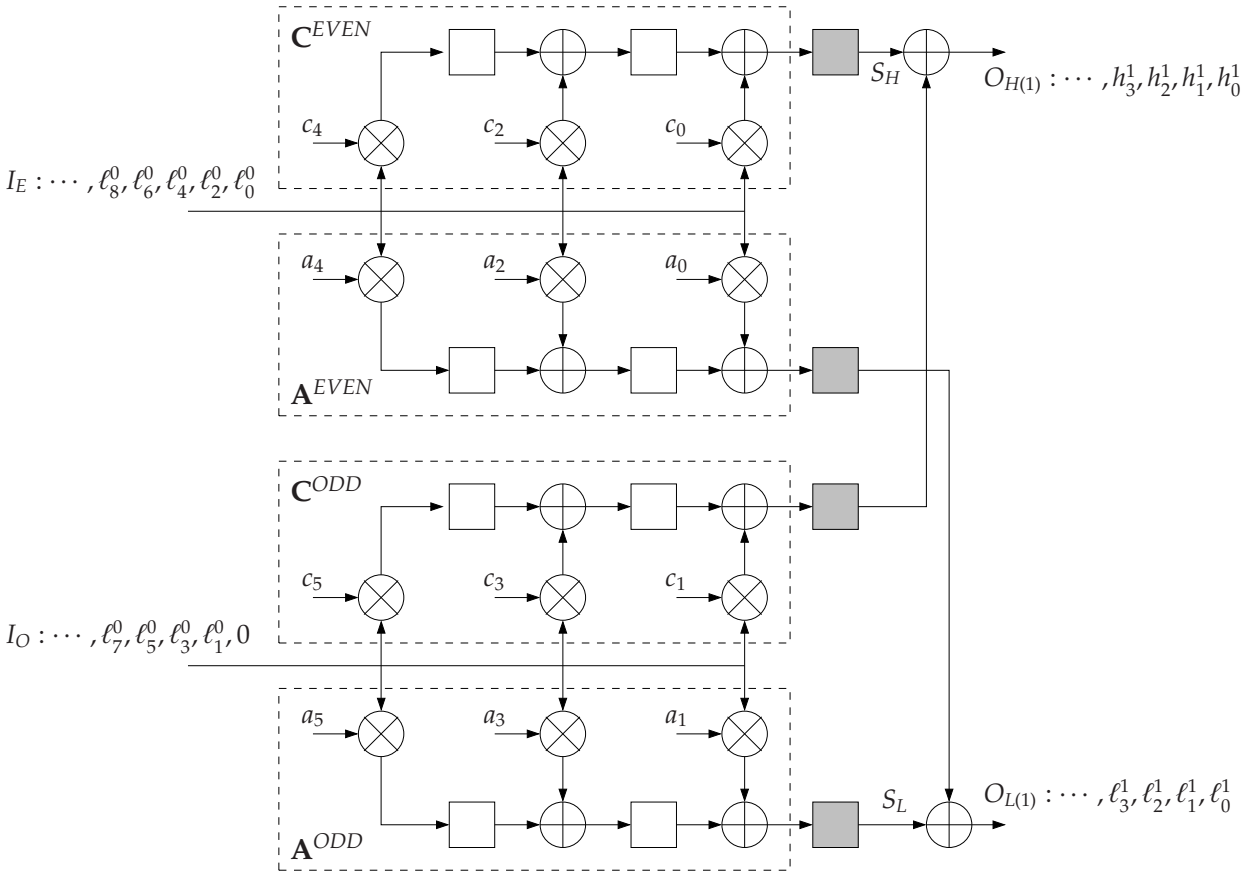


Fig. 3.3: Arc$_J$ architecture for first decomposition level [?].

The block B_1 has two input lines I_E and I_O. I_E feeds ℓ^{0-EVEN} both into $\mathbf{A}^{EVEN}$ and $\mathbf{C}^{EVEN}$, while I_O feeds ℓ^{0-ODD} both into $\mathbf{A}^{ODD}$ and $\mathbf{C}^{ODD}$. Because of the independence of the computations, ℓ_i^{0-EVEN} and ℓ_i^{0-ODD} are fed in parallel during the same clock cycle. Therefore, the input sequence ℓ^0 can be sampled at a frequency twice higher than the working frequency of the device. Four latches at the input of the adders S_L and S_H have no functional reason but have been inserted in order to make the critical paths limited to one multiplier and one adder. By this way, the assumption that the clock period is bounded by the latency of one multiplier and one adder is satisfied. Additional latches inserted at the output of each multiplier could further decrease the minimum allowed clock period. The design of B_J is fully scalable with any value of L and has 100 % efficiency.

In order to perform the second level of decomposition, the subband ℓ^1 produced by block B_1 has to be further transformed. This requires a second block B_2, which has to be pipelined to the output line $O_{L(1)}$ coming from B_1. As previously stated, B_2 has to be provided with L PEs w_i in order to constitute a balanced team with B_1. Therefore, a possible way of employing these PEs is to assign to w_i the computations related both to a_i and c_i. **Fig**. 3.4 depicts the scheme of the second decomposition architecture B_2.

The line $O_{L(1)}$ outputs the samples of ℓ^1 in a sequential data stream at a rate of one sample per clock cycle. Therefore, by splitting $O_{L(1)}$ into two different lines I_E and I_O

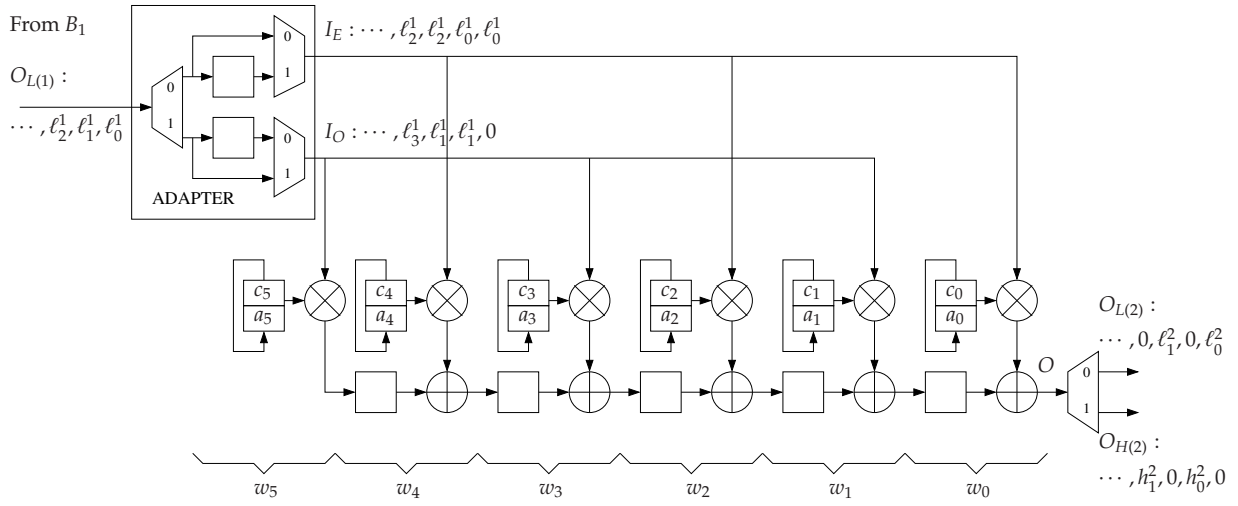


Fig. 3.4: Arc$_J$ architecture for second decomposition level [?].

carrying ℓ^{1-EVEN} and ℓ^{1-ODD} into B_2, we have the possibility of duplication on I_E and I_O before a new sample is sent by $O_{L(1)}$. To perform this, and *adapter* such as shown in **Fig**. 3.4 has to be inserted between $O_{L(1)}$ and the lines I_E/I_O. Block B_2 is also balanced with B_1 since its design has been dimensioned on L PEs. The design on the next higher blocks B_j follows the same principle to balance with the previous block B_{j-1}. Therefore the architecture can deliver maximum performance in terms of sample processing.

Compared to the previous detailed systolic architecture, the generation of the scheduling table for Arc$_J$ is much simpler. Additionally, the interconnects between registers that hold the low-pass and high-pass coefficients and also the samples are straightforward. Instead of merging the multilevel decomposition feature into one single design, as it is the case in the first architecture, Arc$_J$ tries to split each decomposition stage and divides the task on each stage. Knowing that the resulting wavelet decomposition halves the size of the sample at each decomposition level, Arc$_J$ exploits some optimization techniques in the design for the second decomposition stage and so on in order to reduce the number of multipliers and adders. This architecture is more modular and more extensible. One main drawback of Arc$_J$ is that we need to determine during the design time how many decomposition levels we want to have. Once the architecture is fabricated, it is not possible to extend the architecture to make it feasible to compute higher decomposition level. Additionally, we need to determine at the beginning how many taps will be involved in order to distribute and balance all the decomposition blocks. Therefore, wavelet filters that can be exercised by this architecture are also limited to have at maximum the number of taps the architecture provides. Lastly, Arc$_J$ is only suitable to compute fixed number of 1D DWT decomposition. The architecture does not deliver any performance gain when it is used to decompose N-dimensional signals. It happens because in multidimensional transforms, the decomposition is computed one level each for every dimension before the next decomposition stage is carried out.

3.2 Lifting-Based Architectures

Contrary to the filter approach, which separates the signal into low and high frequency components and performs the decimation on both signals afterwards, the second generation of wavelets reduces the computation by performing the decimation in advance. The second generation of wavelets, more popular under the name of lifting scheme, was introduced by Sweldens [?]. The basic principle of lifting scheme is to factorize the wavelet filter into alternating upper and lower triangular 2×2 matrix.

Lifting-based architectures become very popular because of the efficiency these architectures deliver. As the decimation is performed in advance in the lifting scheme, it can be expected that lifting-based DWT architectures are twice efficient compared to the filter-based when dyadic DWT is concerned. Before we discuss in details about the existing lifting-based DWT architectures, it is important to note that lifting scheme is heavily used to compute DWT in JPEG2000. In the transform block of JPEG2000, two different wavelet filters can be applied depending on the compression methods. For lossless compression, (5,3) filter is used because it is simple and requires no scaling, whereas for lossy compression, (9,7) filter is used because it can exploit the locality property of the signal, image in this case, better compared to (5,3) filter.

The polyphase representation of (5,3) decomposition filter can be written as

$$\mathbf{P}_{(5,3)}(z) = \begin{bmatrix} 1 & \frac{1}{4}(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}(1+z^{-1}) & 1 \end{bmatrix} \tag{3.1}$$

and the polyphase representation of (9,7) decomposition filter as

$$\mathbf{P}_{(9,7)}(z) = \begin{bmatrix} \frac{4\sqrt{2}}{5} & 0 \\ 0 & \frac{5\sqrt{2}}{8} \end{bmatrix} \begin{bmatrix} 1 & \frac{15}{16}(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{4}{5}(1+z) & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{16}(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{3}{2}(1+z) & 1 \end{bmatrix} \tag{3.2}$$

Note that because solving the polyphase representation of a wavelet filter is not unique, it is possible to have different representations for the same wavelet filter.

From both equations, it is obvious that the prediction and the update functions are very straightforward. Without loss of generality, the predictor $\mathscr{P}$ and the updater $\mathscr{U}$ for these types of filters can be written as

$$\mathscr{F} = c(1 + z^{\pm 1}) \tag{3.3}$$

with c as a coefficient. We can see that the predictor or the updater consumes only one unit delay, one adder, and one multiplier. The prediction or the update matrix requires one additional adder in order to update the odd or even sample. Based on this assumption, there exist quite a lot of architectures that deal with DWT decomposition and reconstruction based on lifting scheme representations. This is however not always the case to have such simple forms for the predictor or the updater, especially for another class of wavelet filters such as Daubechies, Symlet, and Coiflet wavelet filters. We will discuss more on

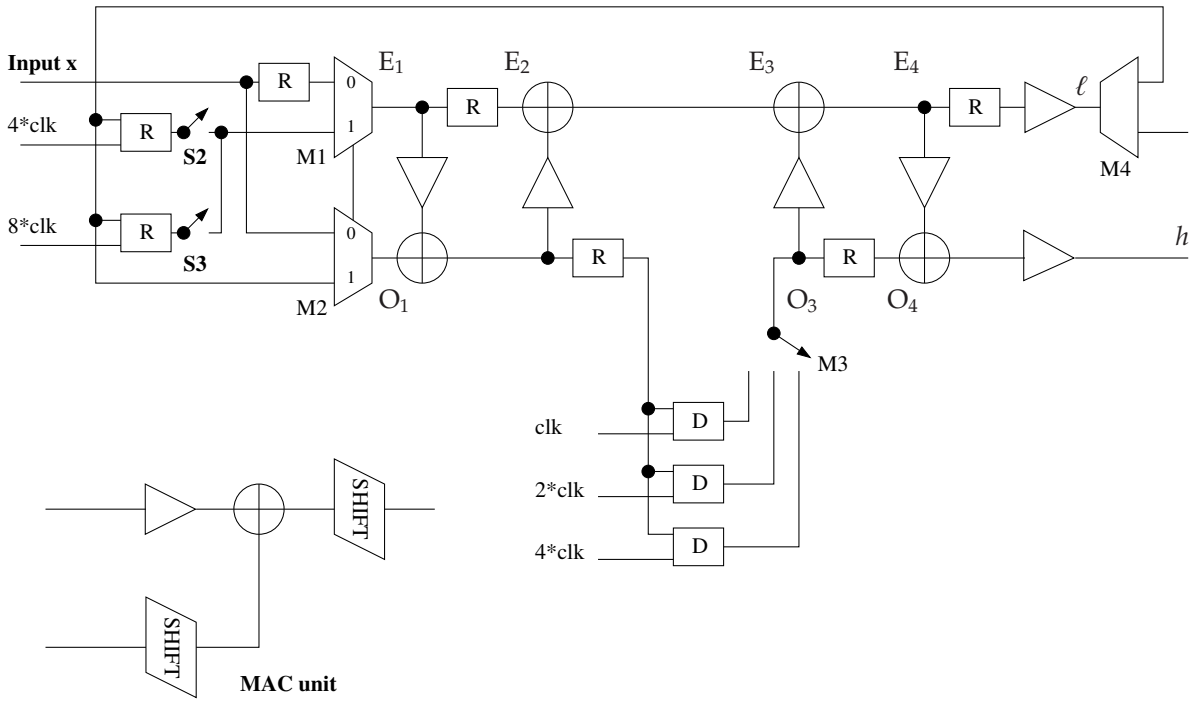


Fig. 3.5: Recursive architecture proposed by Liao [?].

this topic in **Chap**. 4. Several techniques in arithmetic sharing, register allocations, reducing switching activities, hybrid architectures, etc. are developed [?, **?**, ?, ?, ?, ?, **?**, ?, **?**, **?**] to optimize the lifting-based DWT for JPEG2000 and to adapt their hardware realizations efficiently.

3.2.1 Recursive Architecture

Contrary to filter-based approach, there are quite few lifting-based architectures that are not targeted for JPEG2000 DWT. Therefore, before we come into more recent developed architectures which are mainly dedicated to compute (5,3) and (9,7) wavelet filters for 2D image, we discuss one of the lifting-based architectures proposed by Liao *et al.* [?]. The architecture is called recursive architecture.

The recursive architecture (RA), depicted in **Fig**. 3.5, is composed of basic building circuits such as delay units, MAC, and multipliers. The MAC unit consists of a multiplier, an adder, and two shifters. The function of the shifters is to scale the partial results so that the accuracy can be better preserved. Instead of exploiting periodicity or mirror extension, RA uses zero padding to simplify the controller. Therefore, except for Haar wavelet filter, the number of samples increases in every decomposition. The architecture shown in **Fig**. 3.5 can calculate three stages of Daub-4 DWT. Multiplexers M1 and M2 select the input data: when select signal S1=0, the RA computes the first-stage DWT; when S1=1, the RA computes the second and higher stages. Switches S2 and S3 select the data for computing the second and third stages. The registers in this region store the

even samples, so that they can be synchronized with the odd samples. The multiplexer M3 selects the delayed signal for each stage. The remaining registers synchronize the inputs and the outputs of the MACs. In order to save registers, the delay registers and the input registers are controlled by different clock signals, whose frequencies are specified in the figure. If extra storage is not a concern, these registers can also be controlled by the system clock.

The control signals for the switches in a RA can be deduced from the data flow in a relatively straightforward manner. The control signal for the low-pass component is the same as the input switch for the highest stage, and the timing for switch M3 can also be achieved by adding delays (from E1 to E3). The ideal utilization of this architecture approaches 87.5 %.

The main drawback of this architecture is that it requires different clock sources to compute higher decomposition level in order to save the registers. The MAC unit utilizes only one multiplier and one adder. This means that all lifting representations must have one prediction or update term at maximum, i.e. it must be factored as cz^{-n}. Although it is possible for a wavelet filter to have such lifting representation, it will contribute to a longer lifting scheme. More about this topic will be discussed in **Chap**. 4. In addition to the limited number of multiplier and adder, RA provides a fixed number of MACs. Therefore, combined with the previous constraint, only very limited wavelet filters can be used as the DWT kernel in this architecture. Similar to the previous architectures which are based on filter banks, the RA can only be designed to perform a fixed number of decomposition levels, i.e. 3 levels in this case. It is necessary to redesign the architecture and the scheduling in order to let RA perform higher decomposition levels. Although the reconstruction via lifting scheme is straightforward, it is obvious that the RA can only perform DWT decomposition but not reconstruction. This is mainly caused by the recursive topology this architecture employs, which does not allow the reconstruction to use the same architecture in an easy manner.

3.2.2 1D Folded Architecture

A folded reconfigurable 1D DWT core is proposed by Lian *et al.* [?]. It is basically an improvement of an unfolded architecture proposed in [**?**] in order to compute both (5,3) and (9,7) wavelet filters in more efficient way. The architecture is depicted in **Fig**. 3.6.

The architecture works under the assumption that only single read port and write port memory is available, and only single-phase clock signal is used for the system, data read form memory (one per cycle), and write back (also one per cycle). In the split phase of lifting scheme, the data are fed into two shift registers and two samples are read into the predict stage every other cycles. At the output, two output data are available in every other cycle and a parallel to serial circuit is also added for the constraints on single write port memory. That means the input and output data rate of the DWT core are both one sample per clock cycle.

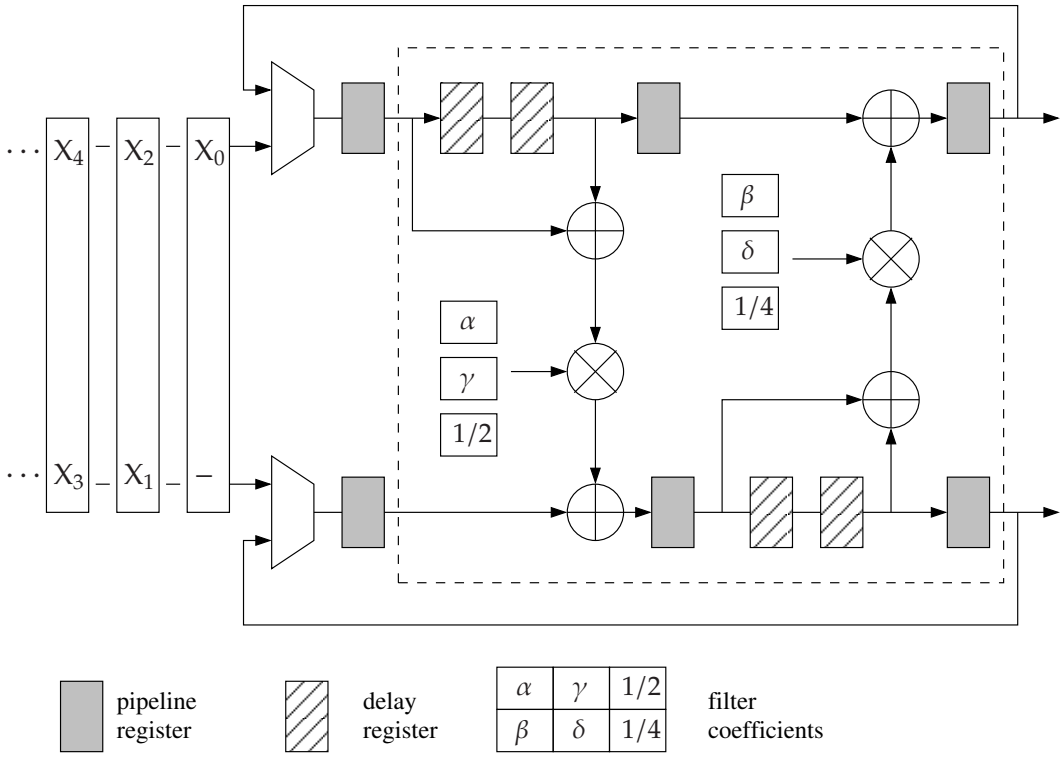


Fig. 3.6: 1D folded architecture proposed by Lian [?].

In the (9,7) filter mode, there are two stages of predict and update operations. Data after the first stage computation are fed back or folded to the input multiplexers for the second stage computation. The computation of the first stage and the second stage are interleaved. The hardware utilization is 100 %. While in the (5,3) filter mode, no folded computing is necessary since there is only one stage for lifting-based operation for this filter. Another difference is that the multiplication in (5,3) filter is in fact only shift-right operation. More specifically, since the filter coefficients are fixed for JPEG2000, the number of bits to be shifted right is also a constant and only hardwired shifting with sign bit extension is necessary. The computation load in (5,3) is much lower than in (9,7). Also, since no interleaving computation of two stages exists in (5,3) mode, the computation time in predict and update phase can be equivalently two times of the clock period. Therefore, the pipeline registers can be bypassed in (5,3) filter mode, with the effect that the latency is reduced without increasing the clock frequency.

Because the architecture is dedicated for JPEG2000, the filter coefficients are fixed during the design. Multiplications can therefore be further optimized. Hardwired multipliers are used instead of real multipliers to achieve a more compact design. The finite-precision coefficients are chosen to be within reasonable error range, as defined in [**?**].

To extend the 1D DWT core to compute 2D DWT in JPEG2000, two cases are considered in this architecture. First case is when a frame memory is necessary and has already existed before DWT operation. The data of the whole image are assumed to be stored in the memory. Although 2D DWT can be scheduled to calculate all rows first (horizontal 1D

DWT), and then all columns (vertical 1D DWT), it is possible to start the Embedded Block Coding with Optimized Truncation (EBCOT) computation once there is a complete code block data available. Due to the in-place computing capability of lifting scheme, the original samples can be replaced directly by the calculated coefficients. Hence, the original frame-size memory is enough. The advantage of this implementation is the ease of data flow control. Due to the interleaving characteristics of the output, i.e. one low-pass sample followed by one high-pass sample, the interleaving storage arrangement will have L-H-L-H form. The second case is when a frame memory is not available or not allowed due to constraint on the cost of the memory size. Then, the concept of line-based DWT [**?**] can be adopted. Since EBCOT is not line-based, the height of the line buffer will depend on the height of the code block. The required buffer size for DWT will be smaller than the frame memory. However, another memory space for the compressed sub-bitstreams of every code block is necessary.

Although this architecture puts (5,3) and (9,7) wavelet filters together, its hardware utilization is actually half of the reported values. The first assumption that this architecture bases, i.e. single memory port, is not optimized in the design. We can obviously see that the core processes one sample per clock cycle, or better said, two samples per two clock cycles. We can also see that both multipliers and adders process two inputs per clock cycle. Knowing this in advance, this architecture does not exploit the time-sharing property of the multipliers and the adders. Therefore, during half of each cycle, these arithmetics are in idle state.

3.2.3 Row and Column Processors Architectures

The third and most popular architecture for computing 2D DWT transform for JPEG2000 composes of two processors, one for performing row transforms and the other for column transforms. Dillen *et al.* combine line-based architecture for computing (5,3) and (9,7) wavelet filters [?]. Their architecture is targeted for FPGA where only one read and one write cycle can be executed at one time in order to keep the routing cost low and the resource utilization minimum. It means that the two samples are processed every two clock cycles, similar to the folded architecture described earlier. Andra *et al.* detail the VLSI implementation of the combined architecture for row and column processors [?]. Whilst each processor has capability to perform only two lifting steps, they utilize the column processor to compute the row transforms for the second lifting part of (9,7) filter. Thus, they can minimize the area. Barua *et al.* details the use of ASAP scheduling for the column processor in order to minimize the temporary memory storage [?, ?]. Wu and Lin further optimize the memory allocation of the combined (5,3) and (9,7) architecture by rescheduling the memory access, especially for the column transforms [?]. Seo and Kim detail the VLSI architecture of line-based lifting for motion JPEG2000 [?, ?]. These architectures inherit similarities, not only in the design of row and column processors, but also in the complexity. One key point is that by reducing the memory size, the complexity

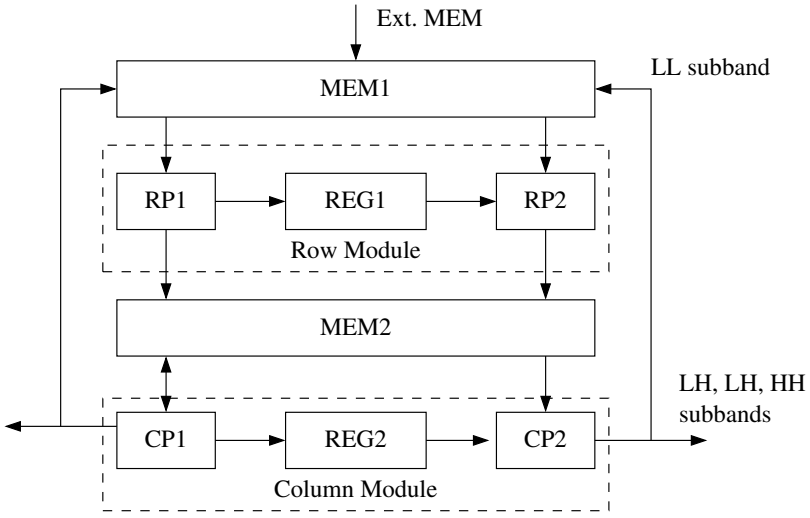


Fig. 3.7: Row and column architecture [?].

to perform the column transforms raises. Several architectures try to relax the complexity by exercising extra storage, while others offer a minimal memory footprint in trade off with high and complex interconnect structures.

Instead of discussing all the existing 2D DWT processors that are based on this architecture, we detail one of the contributions given in [?]. The architecture calculates the DWT and IDWT in row-column fashion on a block of data of size. To perform the DWT, the architecture reads in the block of data, carries out the transform, and outputs the LH, HL, and HH data at each level of decomposition. The LL data is used for the next level of decomposition. To perform the IDWT, all the sub-bands from the lowest level are read in. At the end of the inverse transform, the LL values of the next higher level are obtained. The transform values of the three subbands (LH, HL, and HH) are read in, and the IDWT is carried out on the new data set.

The architecture, as shown in **Fig**. 3.7, consists of a row module (two row processors RP1 and RP2 along with a register file REG1), a column module (two column processors CP1, CP2 and a register file REG2), and two memory modules (MEM1, MEM2). As mentioned earlier, DWT and IDWT are symmetrical if the lifting scheme is used. Hence, we discuss all the details in terms of DWT, as an extension to IDWT is straightforward.

In the (5,3) mode, processors RP1 and RP2 read the data from MEM1, perform the DWT along the rows, and write the data into MEM2. Processor CP1 reads the data from MEM2, performs the column wise DWT along alternate rows, and writes the HH and LH subbands into MEM2 and Ext. MEM. Processor CP2 reads the data from MEM2, performs the column-wise DWT along the rows on which the CP1 did not work, and writes LL subband to MEM1 and HL sub-band to Ext. MEM. The data flow is shown in **Fig**. 3.8.

In the (9,7) mode, there are two passes with transform along one dimension being calculated in a pass. In the first pass, RP1 and RP2 read in the data from MEM1, execute the first two matrix multiplications, and write the result into MEM2. CP1 and CP2 execute

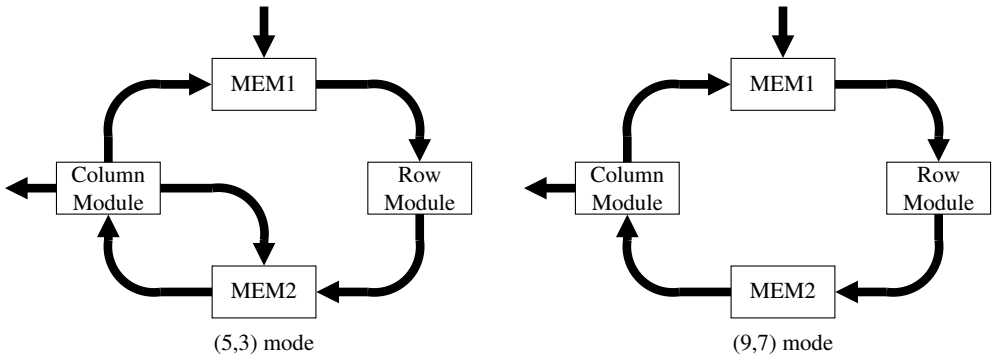


Fig. 3.8: Data flow for (5,3) and (9,7) filter mode [?].

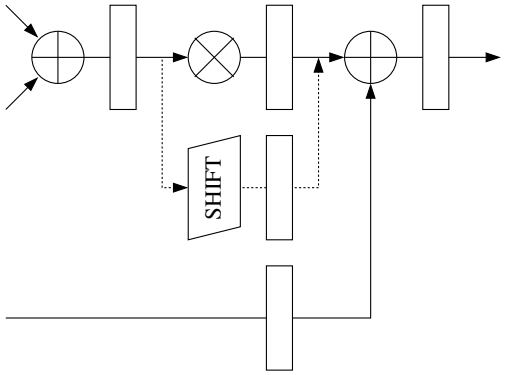


Fig. 3.9: Basic architecture of each processor [?].

the next two matrix multiplications and write results (high-pass and low-pass terms along the rows) to MEM2. This finishes the transform along rows. In the second pass, the transform is calculated along columns. At the end of the second pass, CP1 writes HH and LH sub-bands to Ext. MEM, whereas CP2 writes the LL sub-band to MEM1 and the HL sub-band to Ext. MEM.

The processor consists of two adders, one multiplier, and one shifter, as depicted in **Fig**. 3.9. In order to carry out the scaling step, a shifter is connected to the output of the RP1 and RP2 processors, and a multiplier/shifter is connected to the output of the CP1 and CP2 processors. The MEM1 module consists of two banks and MEM2 module consists of four banks. All the banks have one read and one write port. Further, this architecture assumes that two accesses/cycle are possible. The memory banks in the MEM1 module read in the whole block in the beginning during the forward transform and read in the whole block at the last level during the inverse transform. Therefore, the size of the memory banks in the MEM1 module is $N \times N/2$ each and the size of memory banks in the MEM2 module is N each.

The scheduling for both (5,3) and (9,7) filters is generated by hand and it is later fixed during the design. This means that even though the architecture can compute DWT with two or four lifting steps, we cannot easily change the lifting coefficients without rescheduling. Therefore, this architecture can only perform DWTs with the wavelet filters

assigned during the design time. Because the memory modules are tightly coupled with the processors, it is not easy to extend the architecture to perform higher dimensional DWTs.

3.3 Architectures for Computing Discrete Wavelet Packet

Contrary to the DWT, most of the Discrete Wavelet Packet (DWP) computings are based on software and used to develop better algorithms for specific applications. Yang and Xu develop an image coding algorithm based on a rate-distortion optimized wavelet packet subbands in order to replace the zerotree quantization methods which are not suitable for DWP [**?**]. Carnero and Drygajlo use wavelet packet for perceptual speech coding and enhancement [?]. Image denoising algorithm via best wavelet packet base using Wiener cost function is develop by Shui *et al.* [**?**]. Wavelet packet basis expansion is also used to identify a time-varying nonlinear system [**?**]. Image fusion algorithm based on wavelet packet is developed targeting parallel computing platform [**?**] to achieve real-time constraint.

The developed hardware realizations of the DWP are mainly based on filter banks approach [**?**, **?**, **?**], which are similar to the DWT architectures discussed at the beginning of this chapter. Nonetheless, few lifting-based architectures exist. A folded parallel architecture for lifting-based DWP is presented by Arguello [**?**]. It consists of a group of PEs operating in parallel on the data prestored in a memory bank. Aroutchelvame and Raahemifar [**?**] develop an architecture using a single PE to perform one level of DWP at a time. The main drawbacks of these architectures are that they all use on-chip memory to store the intermediate coefficients and involve intense memory access during the computation, which consume large silicon area and lead to significant power dissipation. Paya *et al.* [**?**] present a recursive pyramid algorithm (RPA)-based folded architecture for computing lifting-based multilevel DWP without any on-chip memory access. However, the scheduling and control complexity is high, which also introduce large numbers of switches, multiplexers and control signals. The architecture is not regular and need to be modified for different number of level of DWP computation.

Double-path delay commutator (DDC) achitecture to compute DWP is proposed by Wang and Gan [?]. **Fig**. 3.10 depicts the scheme of this architecture. The principle is similar to the 1D folded architecture for JPEG2000 [?]. At each level in the DDC architecture, multiple groups of butterfly operations are mapped on one Daub-4 filter. Contrary to the rest of the lifting steps, the first lifting update of Daub-4, i.e. $d^{(1)}(k)$, requires two even samples. Thus, 2 PEs are required to compute this function. The architecture has only 50 % utilization, or in other words, it can accept only one input per clock cycle. That is why the delay registers, depicted as nD in the figure, are two-level depth. Because this architecture can only compute one decomposition level, another group of butterfly operations is required in order to compute the second decomposition level, and so on.

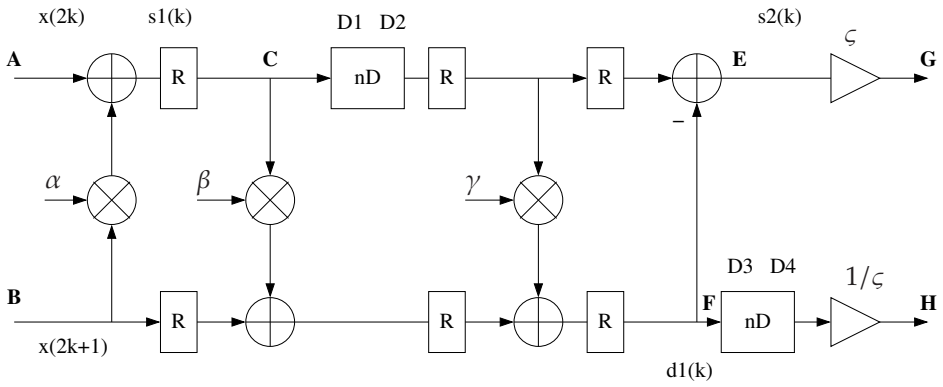


Fig. 3.10: Folded architecture for DWP [?].

We can directly see that this architecture is suitable for computing Daub-4 filter. If different filters are used, the architecture and its interconnects need to be redesigned. Because of the constrained butterfly operations, different wavelet filters cannot be easily used, even they have the same lifting length as Daub-4. Additionally, DDC architecture exploits zero-padding in order to compute the transform in the boundary regions. This approach increases the sample size during each transform in order to make it reversible. If we apply this rule to the DWP where all existing bands need to be decomposed, the resulting size of the final transform will increase significantly. Finally, because of lack of the memory, DDC architecture considers only data stream as inputs. It does not have any memory interface to control how the samples are fed and stored, which are the main issues in DWP. Therefore, although DDC architecture is capable of computing DWP, basically, it is only stream PEs for computing DWT.

3.4 Concluding Remarks

We have discussed several hardware architectures to compute wavelet transforms. Two different approaches were detailed, i.e. the traditional approaches using filter banks and the more recent and efficient approaches using lifting schemes. Several architectures are designed to compute wavelet transform using specific wavelet filter and the others can be tailored during the design-time to adapt with different wavelet filter. Some architectures extend the flexibility to exercise the wavelet filter chosen during run-time. Nevertheless, these architectures offer very limited supports regarding the wavelet filters used during the transforms. Therefore, only a small number of wavelet filters can be used. Most architectures have capability to perform DWT decomposition as well as DWT reconstruction. Nonetheless, there exist quite few architectures that are able to compute DWP and in our knowledge, we do not find any architecture that combines both DWT and DWP decomposition and reconstruction into one design.

Chapter 4

Architecture Design Consideration

Contents

4.1 Algorithm and Application Mapping 59

4.2 Observation on the Algorithm for Discrete Wavelet Decomposition and Reconstruction 64

4.2.1 Observation on the Lifting Scheme 64

4.2.2 Wavelet Transform and Wavelet Packet 70

4.3 Design Analysis 71

4.4 Fixed-Point based PE 72

4.4.1 Resource-aware Fixed-Point PE 73

4.4.2 High-Performance Fixed-Point PE 76

4.5 The Need of Floating-Point 79

4.5.1 Standard IEEE 754 Format 81

4.5.2 Floating-Point Multiplier 81

4.5.3 Floating-Point Addition Algorithm 84

4.5.4 4-Stage Floating-Point Adder with Three Inputs 85

4.5.5 3-Stage Floating-Point Adder with Three Inputs 89

4.5.6 Resource-aware Floating-Point PE 92

4.5.7 High-Performance Floating-Point PE 94

4.6 Context Switch for PE 96

4.7 Common Components 100

4.7.1 Main FSM 101

4.7.2 Config 104

4.7.3 Memory 105

4.7.4 Arbiter 108

4.7.5 Source and Sink 108
4.7.6 Latency Counter 110
4.8 Details of the Transform Process 110
4.8.1 1D Discrete Wavelet Decomposition for DWT 111
4.8.2 1D Discrete Wavelet Reconstruction for DWT 114
4.8.3 1D Discrete Wavelet Decomposition for DWP 116
4.8.4 1D Discrete Wavelet Reconstruction for DWP 119
4.8.5 *N*-Dimensional DWT Decomposition and Reconstruction 120
4.8.6 *N*-Dimensional DWP Decomposition and Reconstruction 122
4.9 Concluding Remarks 123

This chapter details the architecture of our proposed wavelet processors. In order to provide a generalized architecture to compute DWT decomposition and reconstruction, as well as DWP decomposition and reconstruction with various wavelet filters, we examine the requirement to build such structure by developing an algorithm that maps the computations of wavelet transform efficiently. The algorithm is detailed in **Sec**. 4.1. **Sec**. 4.2 continues the discussion by applying the algorithm to the lifting scheme. Additionally, the technique to support both DWT and DWP is also covered. Design analysis of wavelet processors is described in **Sec**. 4.3. Four different wavelet processors are developed. Based on their arithmetics, these processors are divided into two categories, i.e. fixed-point and floating-point wavelet processors. On each category, two different architectures are developed. Resource-aware architecture exploits the time-sharing properties of the arithmetic cores in order to reduce the chip area and has a processing speed of $f/2$. High-performance architecture uses dedicated arithmetic cores to increase the performance and has a processing speed of f. The main components of the wavelet processors, i.e. the processing elements, are detailed in **Sec**. 4.4 and **Sec**. 4.5. **Sec**. 4.6 and **Sec**. 4.7 cover the common components that build up the processors. The details of the transform process of 1D and *N*-dimensional signals are discussed in **Sec**. 4.8. The discussion covers also the transform process of DWPs and an optimization technique to reduce the computation time.

4.1 Algorithm and Application Mapping

The principle of lifting scheme is to factorize the wavelet filter into alternating upper and lower triangular 2×2 matrix, which correspond to the dual and primal lifting steps. Daubechies and Sweldens have shown that the polyphase representation can always be factored into lifting steps by using the Euclidean algorithm to find the greatest common

divisors [?, ?]. Thus the polyphase representation becomes:

$$\mathbf{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{M} \begin{bmatrix} 1 & u_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ p_i(z) & 1 \end{bmatrix} \tag{4.1}$$

where $p_i(z)$ and $u_i(z)$ are the Laurent polynomials and K is the normalization factor. The Euclidean algorithm to find the greatest common divisors is used to solve both polynomials $p_i(z)$ and $u_i(z)$. Because an efficient hardware realization is strictly dependent on the degree of both Laurent polynomials, our goal is to develop an algorithm that restricts the resulting degree of these polynomials.

Given an analysis FIR low-pass filter $\tilde{H}(z)$, we can rewrite it by separating the even and odd powers, i.e.

$$\tilde{H}(z) = \tilde{H}_e(z^2) + z^{-1}\tilde{H}_o(z^2) \tag{4.2}$$

Applying these powers as the initial values for $a_0(z)$ and $b_0(z)$, i.e.

$$a_0(z) = \tilde{H}_e(z^2) \tag{4.3}$$

$$b_0(z) = z^{-1}\tilde{H}_o(z^2) \tag{4.4}$$

we have

$$q_0(z) = a_0(z) \;/\; b_0(z) \tag{4.5}$$

$$r_0(z) = a_0(z) \;\%\; b_0(z) \tag{4.6}$$

as the first quotient and remainder of both polynomials such that

$$a_0(z) = b_0(z)q_0(z) + r_0(z) \tag{4.7}$$

The next iteration continues by setting

$$a_{n+1}(z) = b_n(z) \tag{4.8}$$

$$b_{n+1}(z) = r_n(z) \tag{4.9}$$

with $n \in \mathbb{N}$ and by computing the next quotient and the remainder of the new assigned polynomials,

$$q_n(z) = a_n(z) \;/\; b_n(z) \tag{4.10}$$

$$r_n(z) = a_n(z) \;\%\; b_n(z) \tag{4.11}$$

until $r_N(z) = 0$ is reached.

We develop an algorithm that can find solutions for $p_n(z)$ and $q_n(z)$ in an efficient manner. We drop the subscript to ease the naming. Given two Laurent polynomials $a(z)$ and $b(z)$,

$$a(z) = \sum_{k=k_\ell}^{k_r} a[k]z^k \tag{4.12}$$

$$b(z) = \sum_{\ell=\ell_\ell}^{\ell_r} b[\ell]z^\ell \tag{4.13}$$

The algorithm starts by matching the lowest degree, i.e.

$$c_1 = \frac{a[k_\ell]}{b[\ell_\ell]} \tag{4.14}$$

Thus, we have

$$q(z) = c_1 z^{k_\ell - \ell_\ell} \tag{4.15}$$

$$\begin{aligned} r(z) &= a(z) - q(z)b(z) \\ &= a(z) - c_1 z^{k_\ell - \ell_\ell} b(z) \end{aligned} \tag{4.16}$$

Because we match the lowest degree of $a(z)$, the resulting remainder $r(z)$ will have at most one degree less compared to the Laurent polynomial $a(z)$. This approach however contributes to a higher number of iterations, which also leads to a higher number of prediction or update steps. By matching the second term of the Laurent polynomials additionally, it is possible to reduce the number of resulting quotients. Nevertheless, matching only the next term on the queue of the Laurent polynomial $a(z)$, i.e. $a[k_\ell + 1]z^{k_\ell+1}$, might not always be an optimal solution in respect to the magnitude of the resulting coefficient c_2. We will go back with some examples to show the problem that might come by matching only the next term on the queue later. To overcome this issue, we extend the second matching algorithm by looking also other possible solutions by matching the next higher terms. Thus, we have several options as solutions, i.e.

$$c_2^i = \frac{a[k_\ell + i] - c_1 b[\ell_\ell + i]}{b[\ell_\ell]}, \qquad i = 1, 2, \cdots, I \tag{4.17}$$

and the resulting quotient $q(z)$ becomes

$$q(z) = c_1 z^{k_\ell - \ell_\ell} + c_2^i z^{k_\ell - \ell_\ell + i} \tag{4.18}$$

Finding the best solution c_2^i for the quotient $q(z)$ can be quite challenging. Therefore we need to define some conditions that need to be satisfied by the best among possible solutions.

- Having two factors in quotient $q(z)$, i.e. $c_1 \neq 0$ and $c_2^i \neq 0$, is desired in order to reduce the degree of the resulting remainder $r(z)$ instead of having only one non-zero term, i.e. $c_1 \neq 0$ and $c_2^i = 0$.
- The first on the queue solution, i.e. c_2^1, is desired because we can match and eliminate the first two terms of the Laurent polynomial $a(z)$ directly, which makes the resulting remainder to have Laurent polynomial degree two less compared to $a(z)$.
- Nearest to one coefficient, i.e. $c_2^i \approx 1$ is desired to avoid the underflow and overflow by the arithmetic operations, which can lead to the quality degradation of the transformed signal.

Keeping these constraints in mind, we choose the best solution by sorting the solutions first. Two stage sorts are used to differentiate the cases. The end goal is to select quotient $q(z)$ that consists of two non-zero coefficients if possible and these coefficients do not have very small or very large magnitudes which can lead to numerically unstable solutions.

for $i = 1$ to $I - 1$ **do**
  for $j = i + 1$ to I **do**
    % Basic sorting
    if both c_2^i and c_2^j are zero **then**
      Do not swap
    else if c_2^i is zero and c_2^j is near one **then**
      Swap solutions
    else if c_2^j is zero **then**
      Do not swap
    else
      Perform extended sorting
    end if
    % Extended sorting
    if extended sorting is requested **then**
      % Distance is defined as the degree of the resulting quotient
      if Distance i < Distance j **then**
        % Lower distance has priority.
        % Therefore, swap is only necessary if c_2^j provides much better solution
        if $c_2^j \approx 1$ compared to c_2^i **then**
          Swap solutions
        end if
      else
        if ($c_2^i <$ *min_threshold* and $c_2^j \approx 1$) or ($c_2^i >$ *max_threshold* and $c_2^j \approx 1$) **then**
          Swap solutions
        end if
      end if
    end if
  end for
end for

The output of this solution sorting algorithm is the sorted and best-suited solutions. Using the first sorted solution, we can automate the iterative finding of resulting quotients $q_n(z)$. To give a better insight about how this algorithm works, let us take Daub-12 wavelet filter as an example. Compactly supported wavelets that generate orthonormal bases constructed by Daubechies in [?] is defined as

$$ {}_N m_0(\xi) = \frac{1}{\sqrt{2}} \sum_{n=0}^{2N-1} {}_N h_n e^{-jn\xi} \tag{4.19} $$

Tab. 4.1: The filter coefficients of Daub-12 for the compactly supported wavelet.

n	${}_N h_n$
0	-0.00107730108500
1	0.00477725751101
2	0.00055384220099
3	-0.03158203931803
4	0.02752286553002
5	0.09750160558708
6	-0.12976686756710
7	-0.22626469396517
8	0.31525035170924
9	0.75113390802158
10	0.49462389039839
11	0.11154074335008

and for $N = 6$, the filter coefficients are summarized in **Tab**. 4.1. Note that these coefficients are normalized so that $\sum_n {}_N h_n = \sqrt{2}$.

Splitting the even and odd powers and use them as initial values for a_0 and b_0 gives

$$a_0(z) = 0.4946z^{-5} + 0.3153z^{-4} - 0.1298z^{-3} + 0.0275z^{-2} + 0.0006z^{-1} - 0.0011 \tag{4.20}$$

$$b_0(z) = 0.1115z^{-5} + 0.7511z^{-4} - 0.2263z^{-3} + 0.0975z^{-2} - 0.0316z^{-1} + 0.0048 \tag{4.21}$$

Using the algorithm described earlier, we can constraint the finding of the greatest common divisor by looking only at the closest terms. This leads to

$$\begin{aligned} q_0(z) &= 4.434468z^0 - 27.036122z^1 \\ q_1(z) &= 0.005266z^{-2} + 0.037084z^{-1} \\ q_2(z) &= 21647.782504z^0 + 12217.761781z^1 \\ q_3(z) &= 0.000118z^{-2} - 0.000055z^{-1} \end{aligned}$$

Note that $q_2(z)$ and $q_3(z)$ may lead to the instability during the computation. To overcome the higher dynamic coefficient ranges, we can relax the algorithm to search also the solution on the next closest terms, which gives us

$$\begin{aligned} q_0(z) &= 4.434468z^0 + 7.832086z^2 \\ q_1(z) &= -0.028679z^{-1} - 0.156193z^0 \\ q_2(z) &= 4.011680z^{-1} + 6.380297z^0 \\ q_3(z) &= 79.577955z^{-1} - 9.252550z^0 \\ q_4(z) &= 0.082265z^{-1} + 0.014983z^0 \\ q_5(z) &= -425.866957z^{-1} - 20.234690z^1 \\ q_6(z) &= 0.006740z^0 + 0.000527z^1 \end{aligned}$$

This solution contributes to a longer lifting scheme because at some iteration steps, we match the next closest terms (e.g. solving q_0 and q_5), which only decreases the degree of the resulting remainder by one. Nevertheless, this solution reduces the dynamic coefficient range on the quotients, therefore numerical stable solutions can be achieved without sacrificing the resolution.

There exist different algorithms to compute the Eucledian distance by using the greatest common divisor algorithm. As an example, we can also match the highest terms instead of the lowest ones. The main goal of the algorithm that we develop is to show that the wavelet filters can be factorized into smaller lifting steps that consist of only two terms each. This is a very important issue when we come to the hardware realization of the wavelet transforms. Additionally, matching only two terms on each iteration is relatively straightforward compared to matching with more that two terms. Also, only very limited wavelet filters can be matched with higher terms, as it is the case of several biorthogonal wavelet filters. Nonetheless, these wavelet filters can also be factored into lifting steps that comprise of two terms at maximum.

One final remark regarding the factorization algorithm. We stated that the main purpose of this algorithm is to provide interface to the hardware realization of the wavelet transforms. In order to establish a flexible architecture for wavelet processor that can perform various wavelet transforms using different classes of wavelet filters, it is necessary to investigate at the beginning the possible constraints and also solutions that are both easy to manage and consume less chip area. Therefore, the algorithm described earlier might not deliver an optimal solution for some wavelet filters, that is, the resulting coefficients might have high dynamic ranges. One solution for example, it might also be required to investigate the resulting remainder on each iteration and not only the resulting quotient in order to optimize the better coefficients selection. This is however out of the scope of our discussion.

4.2 Observation on the Algorithm for Discrete Wavelet Decomposition and Reconstruction

4.2.1 Observation on the Lifting Scheme

Rewriting the polyphase representation of the lifting scheme using naming convention for MRA,

$$\begin{bmatrix} S_{j-1}(z) \\ D_{j-1}(z) \end{bmatrix} = \mathbf{P}(z) \begin{bmatrix} S_{e,j}(z) \\ S_{o,j}(z) \end{bmatrix} \tag{4.22}$$

where $\mathbf{P}(z)$ is the polyphase matrix, $s_{e,j}(t) = s_j(2t)$ and $s_{o,j}(t) = s_j(2t-1)$, $s_j(t) \in \mathbf{V_j}$. $s_{j-1} \in \mathbf{V_{j-1}}$ represents the lower resolution, the average, of the MRA space of the signal $s_j(t)$, which is the resulting low-pass signal of the original function $s_j(t)$. $d_{j-1}(t) \in \mathbf{W_{j-1}}$ represents the

details that are extracted from the signal $s_j(t)$, which is the resulting high-pass signal of the original function $s_j(t)$.

The polyphase matrix $\mathbf{P}(z)$ is given by

$$\mathbf{P}(z) = \mathbf{K} \prod_{i=1}^{M} \mathbf{U_i}(z)\mathbf{P_i}(z) \tag{4.23}$$

with the normalization factor $\mathbf{K}$ defined as

$$\mathbf{K} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \tag{4.24}$$

and $\mathbf{P_i}(z)$ and $\mathbf{Q_i}(z)$ as

$$\mathbf{P_i}(z) = \begin{bmatrix} 1 & 0 \\ b_i(z) & 1 \end{bmatrix} \tag{4.25}$$

$$\mathbf{U_i}(z) = \begin{bmatrix} 1 & a_i(z) \\ 0 & 1 \end{bmatrix} \tag{4.26}$$

The 2×2 matrices $\mathbf{P_i}(z)$ and $\mathbf{U_i}(z)$ correspond to the prediction and update steps of the lifting scheme. It is obvious that the prediction and the update occur alternately in the discrete wavelet decomposition based on lifting scheme representation.

Fig. 4.1 shows the block diagram to perform discrete wavelet decomposition with lifting scheme. The lifting-based discrete wavelet decomposition splits the signal into even and odd parts at the first stage. The split signals are processed by an alternating series of predictors and updaters, whereas on some wavelet filters, an updater may come before a predictor. On the final stage, the multiplication with the normalization factor takes place in order to conserve the energy. The predictors $P_i(z)$ and the updaters $U_i(z)$ correspond to the Laurent polynomials $b_i(z)$ and $a_i(z)$ of the alternating triangular matrices $\mathbf{P_i}(z)$ and $\mathbf{U_i}(z)$. The dash boxes in the figure represent an *atom function* of the lifting scheme. Each atom function corresponds to a matrix multiplication either with a prediction matrix $\mathbf{P_i}(z)$ or an update matrix $\mathbf{U_i}(z)$. The atom function of a predictor first predicts the odd samples from the even samples. Because the prediction is not perfect, one can expect errors on the prediction. Thus, instead of outputting the redundant odd samples, one can compute the errors and output them instead. Furthermore, because the predictor does not alter the even samples, the atom function of an updater actualizes the even samples with the values computed by the update block whose input comes from the predictor. This mechanism is repeated until it reaches the normalization stage. The amount of the atom functions depends on the wavelet filter used as the kernel to compute the DWT and on the factorization of the wavelet filter into its lifting scheme representation.

The reconstruction of DWT, as depicted in **Fig**. 4.2, performs exactly everything backwards. It starts with the multiplication with normalization factor, continues with a series

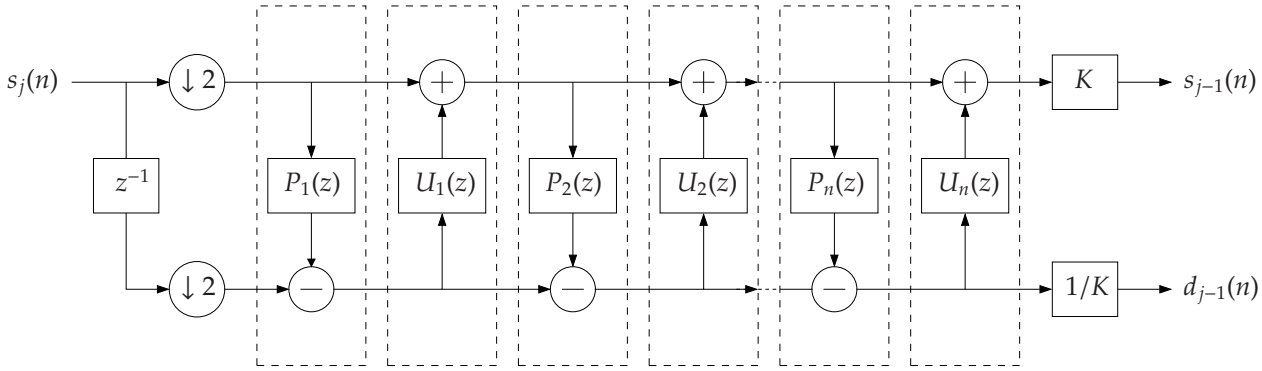


Fig. 4.1: Lifting scheme of discrete wavelet decomposition.

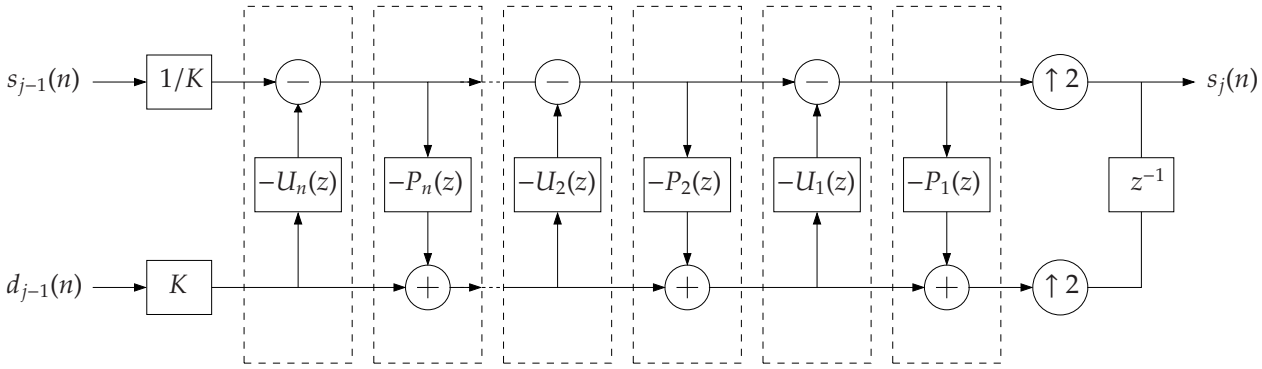


Fig. 4.2: Lifting scheme of discrete wavelet reconstruction.

of updaters and predictors, and finishes with the merging of the outputs. One of the advantages of using lifting scheme to compute the DWT is that it is very simple to construct the reconstruction matrices. We only need to exchange the sign of both predictors $P_i(z)$ and updaters $U_i(z)$.

Another advantage of the lifting scheme, as we can notice from **Fig**. 4.1 and **Fig**. 4.2, is that each prediction or update atom function depends only on the atom function located directly before it, or for the atom function located at first, it depends on the inputs directly. It means that it is not necessary to compute the whole chain of the lifting scheme all in one in order to decompose or reconstruct the signal. Instead, we can split the computation into several smaller steps, store the temporary results, and perform the next part of the lifting scheme. This type of mechanism is called *in-place* computation. This feature becomes very important and also useful in the hardware realization when the cost of the hardware needs to be minimized. We exploit the in-place computation in our design so that wavelet decompositions and also reconstructions that involve higher order wavelet filters, which contribute to longer lifting steps, can also be performed by still maintaining low hardware cost. **Fig**. 4.3 depicts the case. Here, group of atom functions $A_m(z)$, represented with the dashed boxes, consist of several prediction and update atom functions $P_n(z)$ and $U_n(z)$. The in-place computation results are stored in the temporary memory. Therefore, we are able to perform the wavelet decomposition group-wise.

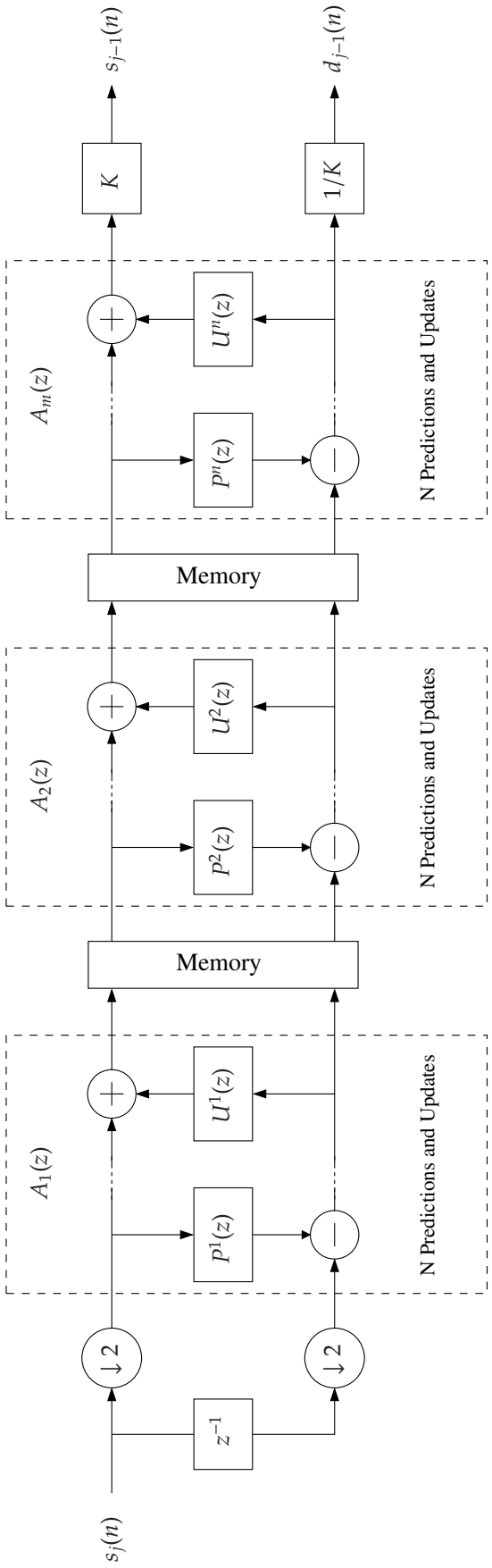


Fig. 4.3: Discrete wavelet decomposition that exercises longer wavelet filter.

Since an atom function of a predictor and an atom function of an updater perform a similar computation (see **Eq**. (4.25) and **Eq**. (4.26)), the hardware architecture for both functions is exactly the same. Taking this into account, we propose novel wavelet processors which are based on N processing elements to cope with N lifting steps. As previously discussed, due to the nature of the lifting scheme, wavelet filters that have longer lifting scheme representations can easily be broken down into smaller lifting steps that the processor can compute (i.e. N lifting steps each). This means that the processor that implements N processing elements is not limited to performing the wavelet transform up to N lifting steps only.

The core behind our proposed architecture is the processing element (PE), which performs the atomic prediction or update. To maximize the performance, the PE utilizes the parallelism by using a pipeline mechanism to guarantee the outputs to be available in every clock cycle. Additionally, we also exploit the hardware cost optimization by making the PE to be time-shared in some designs. This approach reduces the required number of multipliers and the adders on the PE by half in exchange of the performance.

As the lifting scheme breaks a wavelet filter into smaller predictions and updates, the resulting predictor and updater can be limited to have a maximum Laurent polynomial degree of one. Nevertheless, the predictor or the updater of higher order wavelet filters may have the higher factors as well. Recall the discussion on finding the optimal quotient. From **Eq**. (4.18), without loss of generality, we can formulate the predictor or the updater polynomial as:

$$c(z) = c_1 z^{-p} + c_2 z^{-q} \tag{4.27}$$

with c_1 and c_2 as the polynomial constants and $|p - q| \leq N, N \in \mathbb{N}$. This implies that on each stage (either as a predictor or an updater), the PE would perform two multiplications and two additions. As an example, the first predict and update steps of Daub-2 can be written as:

$$\begin{aligned} \begin{bmatrix} s^{(1)} \\ d \end{bmatrix} &= \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ d \end{bmatrix} \\ &= \begin{bmatrix} s + d \cdot \sqrt{3} \\ d \end{bmatrix} \end{aligned} \tag{4.28}$$

$$\begin{aligned} \begin{bmatrix} s^{(1)} \\ d^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4} z^{-1} & 1 \end{bmatrix} \begin{bmatrix} s^{(1)} \\ d \end{bmatrix} \\ &= \begin{bmatrix} s^{(1)} \\ d + s^{(1)} \cdot \frac{-\sqrt{3}}{4} + s^{(1)} \cdot \frac{2-\sqrt{3}}{4} z^{-1} \end{bmatrix} \end{aligned} \tag{4.29}$$

which perform one multiplication and one addition in order to solve $s^{(1)}$ (as shown at the top resulting term in **Eq**. (4.28)) and two multiplications and two additions to solve $d^{(1)}$ (as shown at the bottom resulting term in **Eq**. (4.29)). The naming convention, as used in many textbooks, is depicted in **Fig**. 4.4 to describe each lifting step in an easy manner.

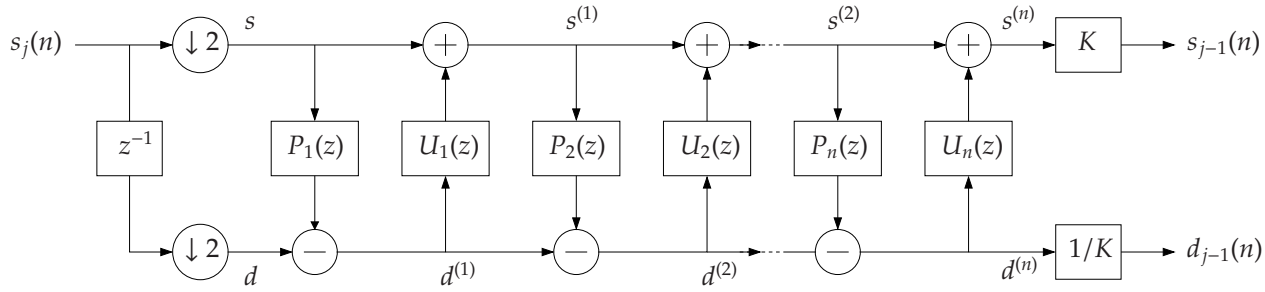


Fig. 4.4: Step by step discrete wavelet decomposition.

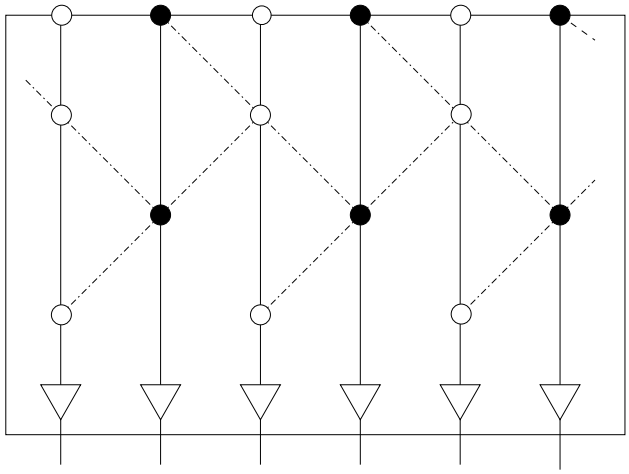


Fig. 4.5: Lattice structure lifting steps of Daubechies-2 wavelet filter.

The term s_j is used instead of $s_j(t)$ for the sake of simplicity. s_j represents the input signal. This signal is split into even and odd components, namely s and d. The first prediction step predicts the odd samples d from the input samples s, resulting $d^{(1)}$. The first update step updates the even samples s from the first prediction result $d^{(1)}$, resulting $s^{(1)}$, and so on. On some liftings, the update may come first before the prediction, as it is the case of the Daub-2 wavelet.

Another representation to describe the data dependencies in lifting schemes is realized by using its lattice structure. **Fig**. 4.5 and **Fig**. 4.6 illustrate the simple lattice structure of Daub-2 and the slightly complex Coiflet-2 lifting-based DWT which are derived from the lifting equations. Here the structured computations which are performed on each stage (either as a predictor or an updater) along with the data dependencies from the previous stages can easily be deduced. Black dots and white dots on the top represent the even and the odd signals whereas the dots in the middle represent the in-place prediction (white) and update (black). The dash-dot line and the triangle represent the multiplication with lifting coefficient and normalization factor respectively. Daub-2 wavelet filter shows its regularity. Coiflet-2 is well-structured on the upper part, whereas on the lower part the predict and update computations require delayed samples (z^{-1} and z^{-2}) and future samples (z^1 and z^2).

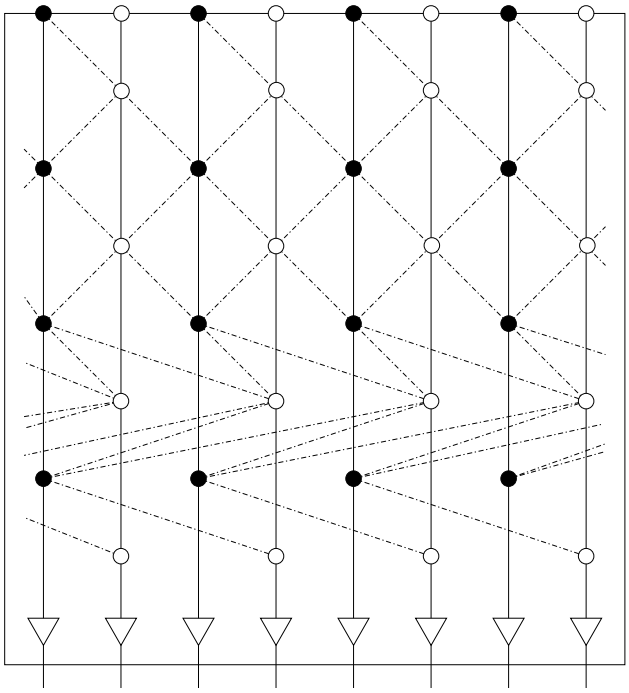


Fig. 4.6: Lattice structure of lifting steps of Coiflet-2 wavelet filter.

4.2.2 Wavelet Transform and Wavelet Packet

DWT is a multiresolution signal analysis. In the traditional DWTs, only the low-pass component is used on the next transform to generate octave-spaced frequency bands of the multiresolution representation of the corresponding signal. In discrete wavelet packets (DWP), both low-pass and high-pass components are analyzed, resulting equally spaced frequency bands. **Fig**. 4.7 and **Fig**. 4.8 depict both schemes. Note that the illustration uses wavelet transforms based on filter banks approach instead of lifting scheme in order to ease understanding the concept for both schemes. It is obvious that DWT requires less computation time compared to DWP, because at each level, the number of samples is decreased by two. Also, the controller that controls the processor to perform DWT decomposition and their reconstruction is straightforward, while the controller to perform DWP decomposition and reconstruction is more complicated due to the fact that the number of frequency bands that need to be processed increases two folds at each transform. As an example, performing four levels wavelet packet on a signal leads to 16 frequency bands whereas performing four levels wavelet transform generates 5 frequency bands.

Not only is the challenge on the controller, the major issue in DWP is that the resulting high-pass components are much smaller than the low-pass components in normal circumstances. Thus performing multi-level DWP using integer arithmetics would make these high-pass components go to zero, which lead to lower achievable signal-to-noise ratio (SNR) values, if it is not carefully performed. Additionally, the challenge in computing the N-dimensional DWP transforms is higher than the N-dimensional DWP transforms because the processing bands increase exponentially.

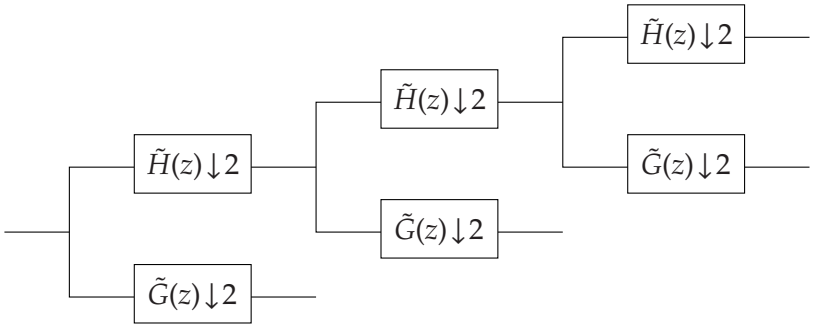


Fig. 4.7: Wavelet transforms using filter banks.

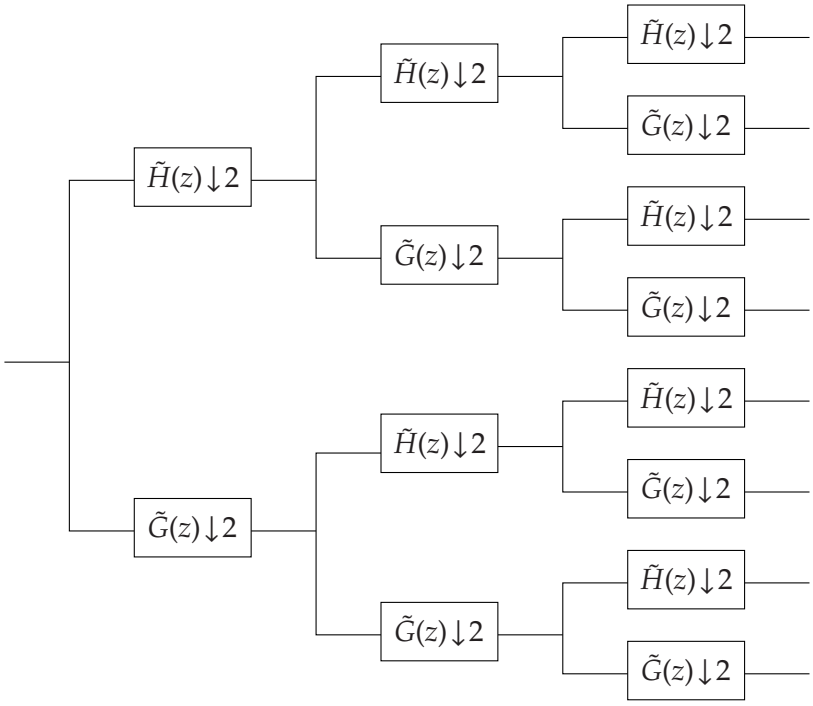


Fig. 4.8: Wavelet packets using filter banks.

Albeit DWT and DWP transforms are different, they inherit similarity. From both figures, it is obvious that each frequency component is decomposed into two components for both cases. We examine this similarity to create a generalized architecture that has a capability to perform DWT decomposition and reconstruction, as well as DWP decomposition and reconstruction.

4.3 Design Analysis

The challenges in providing a generalized architecture for wavelet transforms lie not only on the different types of wavelet filters that can be used on the transforms, but also on the types of the transforms that can be supported, i.e. DWT and/or DWP. In contrary to the hardware realization of the Fourier transform, the number of samples that are used

on wavelet transforms varies on application to application. Therefore, to cover the broad range of applications, we do not limit ourselves to have a fixed memory size. Instead, we let the designers or the applications to choose the size of the memory that can be integrated into our wavelet processor.

In addition to that, some applications require very tight timing constraint in their DSP block and some of them do not. It is well known that hardware cost in term of chip area can be treaded with the delivered performance. We also take into account this philosophy by providing two types of PE architectures, namely resource-aware PEs that utilize time-sharing feature in order to reduce the hardware cost and high-performance PEs that are dedicated for high-performance computing. Furthermore, we also design two different arithmetic cores that can be selected by the designers. Fixed-point arithmetics offer several advantages where they can be realized in hardware with less chip area. The drawback is that these arithmetics are not flexible. One needs to find the best fixed-point representation for his application. In case of wavelet transforms, the representation can vary on filter to filter. Therefore, certain tolerances must be taken into account by providing the worst-case in the fixed-point arithmetics. In contrary, the floating-point arithmetics offer greater flexibility. The designers do not need to care too much about the rollover results introduced by the fixed-point arithmetics, and the number representation is managed in more fashioned way. Of course all of these features do not come for free. Chip area required for floating-point arithmetics is larger than the fixed-point arithmetics and the floating-point arithmetics are more complex in term of their hardware realization.

To cover the above mentioned issues, we propose four types of wavelet processors that are based on the PE that performs the atom function of a prediction or an update. We maintain the same interface definition on the PEs to cope with different number representations, and also different types of demands. Based on the arithmetic cores, we develop two classes of wavelet processors, i.e. fixed-point and floating-point wavelet processors. In each class, two different architectures are developed, i.e. resource-aware architecture that utilizes the time-sharing property of the arithmetic cores and has processing speed of $f/2$ and high-performance architecture that uses dedicated arithmetic cores to boost the performance of the processor and has processing speed of f.

4.4 Fixed-Point based PE

This section describes two different architectures of the PEs that are based on fixed-point arithmetics. The resource-aware PEs utilize one multiplier and one adder in order to perform the atom function, where these arithmetic units are time-shared to reduce the chip size. Additionally, the high-performance PE which is based on the same fixed-point arithmetics will be detail as well.

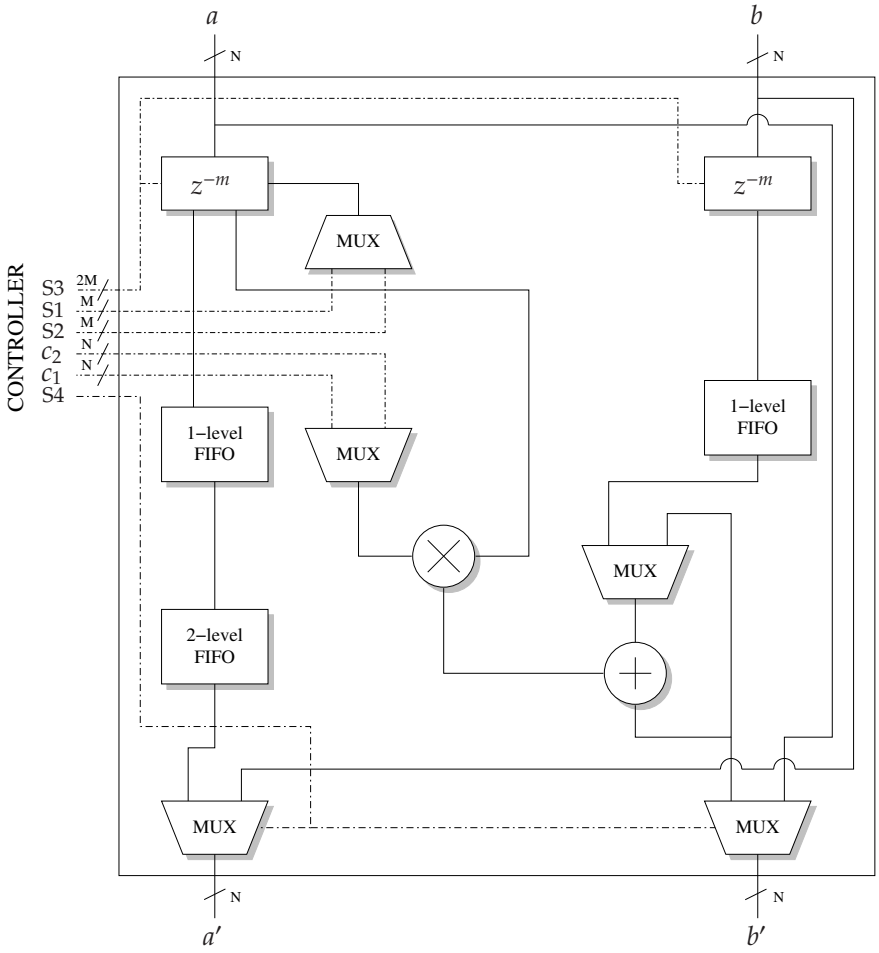


Fig. 4.9: Block diagram of the resource-aware fixed-point PE with one multiplier and one adder.

4.4.1 Resource-aware Fixed-Point PE

Taking into account that multipliers are expensive in term of area and the PE receives two samples (even and odd samples) at once, we have decided to lower the input rate by half. From the performance point of view, the processing rate of the PE will be equal to the processor speed and no longer twice as fast. This also implies that the bottleneck issues on the input and output ports of the PE with the memory will not occur. From the hardware implementation point of view, the PE requires only one multiplier and one adder. This optimization, as detailed later, is accomplished by multiplexing the operands of the multiplier inputs (the *multiplier* and the *multiplicand*) and by feeding the adder result back via the multiplexer.

4.4.1.1 Processing Element

Fig. 4.9 depicts the proposed PE that utilizes one multiplier and one adder [**?**, **?**, **?**]. The PE has two selectors S1 and S2 to choose the prediction or the update samples that correspond to the factors p and q from the Laurent polynomial, resulting two samples m_1

and m_2. Two constants c_1 and c_2 which represent the filter coefficients, are defined and configured by the controller. By delaying the actual samples, selector S3 controls the prediction or the update that requires future samples. Selector S4 is a bypass selector which can be used to bypass the PE. Because lifting steps of the higher order wavelet filters may require distance prediction or update samples, the maximum depth of the unit delay z^{-m}, that determines the maximum delay level, can be freely chosen during the design. The implementation of these unit delays on both port are realized by the configurable FIFO. A FIFO with one-port read interface is needed on port b while a FIFO with two-port read interface is required on port a.

The PE is designed to be a synchronous streaming processing unit in a way that it receives both inputs at the same time, and with some latency, it provides both outputs at the same time. This technique simplifies the design when multiple PE are involved. In other word, the PE is an exact copy of an atom function described earlier.

We use a different naming convention here in **Fig**. 4.9 for input ports a and b, and output ports a' and b', rather than s and d. This is done because the PE can function itself as a prediction atom function or as an update atom function. Thus the terms a, b, a', and b' will interchange themselves based on their function. While the left side of the figure basically consists of configurable and fixed delay units, i.e. $a' = a$, the prediction atom function or the update atom function is processed on the right side of the figure, i.e. $b' = b - \mathscr{P}(a)$ or $b' = b + \mathscr{U}(a)$, depends on the function of the PE. Actually, the subtraction on the atom function of a prediction can be replaced by the addition, and the minus sign is distributed in the prediction itself.

Fig. 4.10 details the MAC (Multiply-and-Accumulate) unit which is implemented inside the PE. Both multiplier unit and adder unit require only one clock cycle to perform their function. c_1 and c_2 correspond to the Laurent polynomial constants, whereas m_1 and m_2 correspond to the outputs of the samples that are selected by S1 and S2. The multiplexer for m_1 and m_2 as a matter of fact does not exist and is drawn here only to illustrate the MAC process. A shifter is utilized as a replacement of the more expensive divider.

The PE is divided into 3 blocks. The first block organizes the input samples from both channels. Organization of both input samples are implemented by using two FIFOs. The depth of the FIFO is determined during the design time. Left FIFO is used to store prediction/update samples A and right FIFO is used to store reference sample B. The second block chooses the inputs of the multiplier and performs the multiplication. As mentioned earlier, the PE utilizes only one multiplier which is time-shared in order to perform two multiplications. The first clock cycle performs the first multiplication (i.e. $c_1 m_1$) and the second cycle performs the second multiplication (i.e. $c_2 m_2$). The third block performs the summation between the reference sample and the prediction/update values. Similar technique is applied here in order to utilize only one adder. As shown in **Fig**. 4.10, the first addition cycle performs

$$o' = b + 2^{-R}(c_1 m_1) \tag{4.30}$$

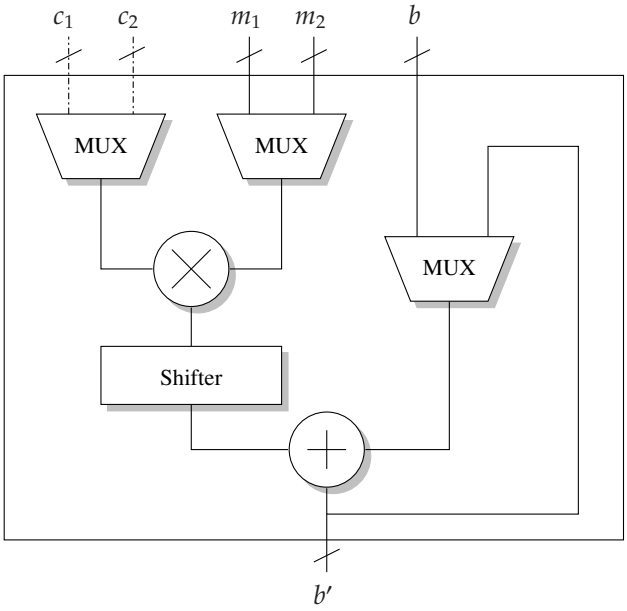


Fig. 4.10: Multiply-And-Accumulate unit.

and the second addition cycle adds-up the first one with $2^{-R}(c_2 m_2)$, i.e.

$$\begin{aligned} b' &= o' + 2^{-R}(c_2 m_2) \\ &= b + 2^{-R}(c_1 m_1 + c_2 m_2) \end{aligned} \tag{4.31}$$

Whilst the input data are integer, the shifter performs the division on the multiplication result with 2^R where R can be freely chosen.

Taking into account that the lifting step coefficients are not integers, a constant shifter is implemented inside the multiplier as a replacement of an expensive divider to perform the fixed-point multiplication. To improve the accuracy, the result of the multiplication is rounded. Because the normal rounding will consume more logic, the rounding is performed *in-place* by the adder. The multiplier outputs one bit after the least significant bit as the fraction bit, and the adder will use this fraction bit as the carry input. This technique not only saves the logic consumption, but also delivers higher performance in term of speed compared to the normal rounding.

Two 1-level FIFOs (First In First Out) are implemented to deal with the multiplier delay and a 2-level FIFO is implemented to compensate the delay which is introduced by the adder. Therefore, both outputs of the atom function are synchronized. This is a very important issue because the PE will be chained with another PE. Thus, by having synchronized outputs, each PE can be treated independently.

One remark from **Fig**. 4.9 is that three multiplexers which control the inputs and also the adder do not have selectors drawn explicitly. This does not mean that these multiplexers do not have selector. Because the PE is designed to be simple, in the way that no finite state machine should be required to control the PE, the selectors do in fact exist to control the data flows. Nonetheless, the selector logic is always synchronized with the

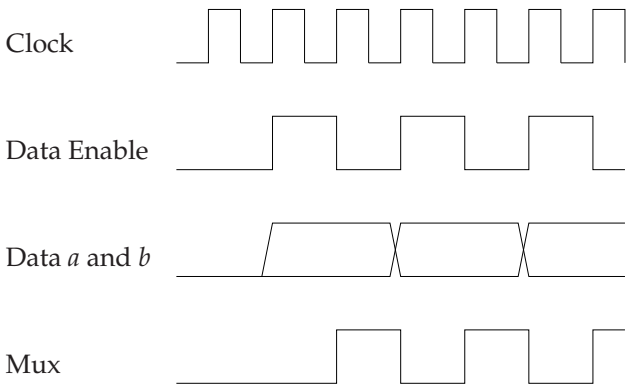


Fig. 4.11: Timing diagram of the multiplexer selectors related to the input samples.

input. **Fig**. 4.11 illustrates the behaviour of the multiplexer selector. Note that because the multiplier and the adder are time-shared, the PE can process the incoming samples as high as two clock cycles. That is why the data enable signal is not always one. This mechanism simplifies the controller of the PE.

4.4.1.2 Normalization

The multiplication with the normalization factor can take place at the end of the transform in case of discrete wavelet decomposition or at the beginning of the transform in case of discrete wavelet reconstruction. As a result, two special processing elements to handle this function are required. Although the normalization step is different compared to the prediction or the update step in a manner that both inputs a and b are multiplied with constants K and $1/K$ respectively, we know certainly that two multiplications take place. To perform this normalization step, we extend the functionality of the PEs located on the top and on the bottom of the proposed wavelet processor instead of implementing a dedicated normalizer unit. Three additional multiplexers are needed to add the normalization factor unit into the PE. **Fig**. 4.12 shows the PE that is used on the top and on the bottom of the proposed architecture. By enabling S5 and setting S1 and S3 to zero, two inputs of the multiplexer before the multiplier correspond to the actual samples a and b (with the normalization factors $K = c_1$ and $1/K = c_2$). The first multiplication product passes through the multiplexer and the 1-level FIFO resulting $a' = Ka$ (the left side). The second multiplication product passes through the multiplexer resulting $b' = b/K$ (the right side). After the first normalization (i.e. $a' = Ka$) takes place, instead of adding a 1-level FIFO on the right output port, the 2-level FIFO is split into two 1-level FIFOs to make both outputs synchronized and to minimize the latency.

4.4.2 High-Performance Fixed-Point PE

In addition to the design of the resource-aware PE that utilizes only one multiplier and one adder, we also develop a PE that satisfies the high-speed signal processing demand.

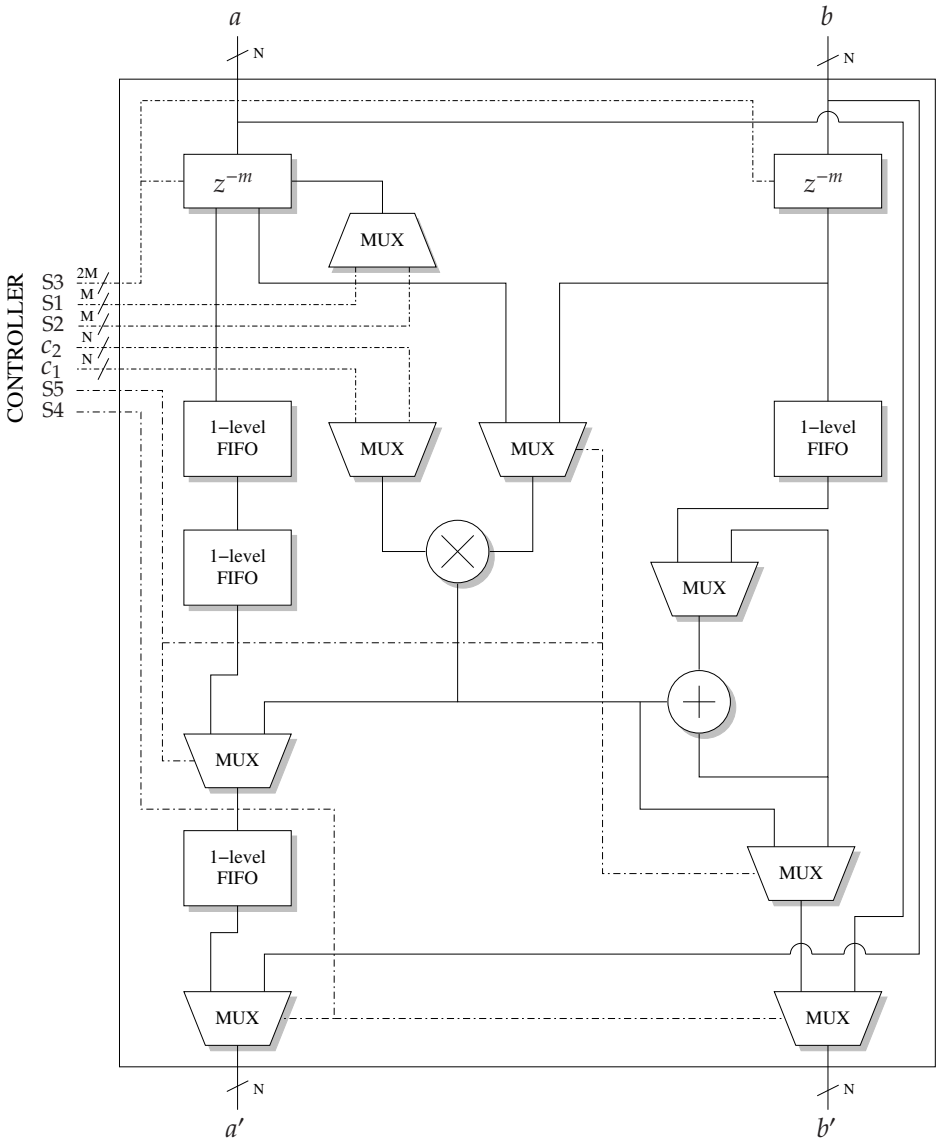


Fig. 4.12: Block diagram of the resource-aware fixed-point PE which is located on the top and on the bottom of wavelet processor.

Nevertheless, we want to maintain the same PE interface as before. In doing so, we do not only simplify the design of the wavelet processor, but we also deliver flexibility on the design itself. As a result, a type of PE can easily be replaced with another type of PE.

4.4.2.1 Processing Element

Fig. 4.13 depicts the proposed PE that employs two multipliers and two adders to increase the throughput [**?**]. The high-performance PE has a pipeline-based architecture in order to maximize the performance. Similar to the resource-aware PE, this PE has two selectors S1 and S2 to choose the prediction or the update samples that correspond to the factors p and q from the Laurent polynomial. Two constants c_1 and c_2 that represent the filter coefficients are defined and configured by the controller. Selector S3 has also the same

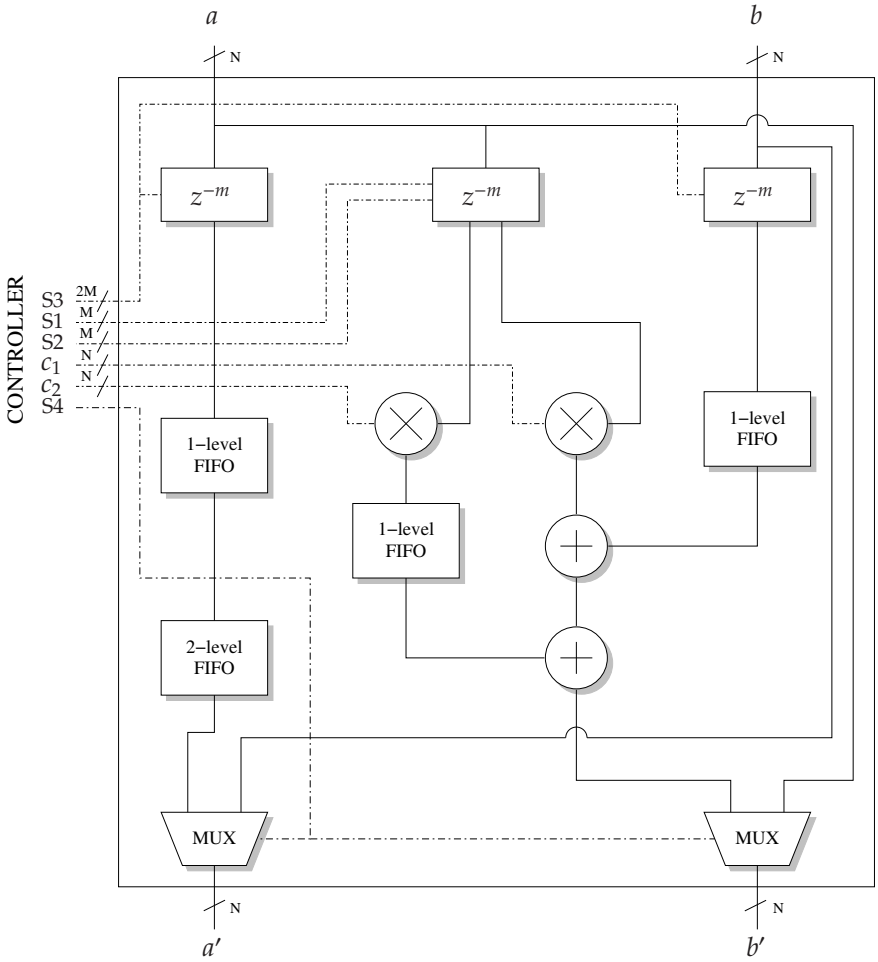


Fig. 4.13: Block diagram of the high-performance fixed-point PE with two multipliers and two adders.

function. By delaying the actual samples, selector S3 controls the prediction or the update that requires future samples. Selector S4 is a bypass selector. From the input and output interfaces, and also the configuration interface, this PE has the same interfaces as the PE that uses only one multiplier and one adder. This is an important design issue of the other components to have a common interface among the PEs. Thus, it simplifies the dependencies on the rest of the components that build up the processor.

The high-performance PE is divided into 3 blocks. The first block organizes the input samples from both channels. The second block performs the multiplication on the samples that are selected by S1 and S2 with the constants c_1 and c_2. Two 1-level FIFOs on both input samples are implemented to compensate the multiplier delay. The last block performs the additions of three values. One 2-level FIFO is implemented to compensate the delay introduced by two adders.

Two significant differences compared to the resource-aware PE can be noticed. This PE requires less multiplexers. The only required multiplexers are the multiplexers that are located at the bottom to bypass the functionality of the PE. Another difference is that this

PE requires additional one-port FIFO along with two-port FIFO used load the samples m_1 and m_2 selected by the selector S1 and S2. We have examined that introducing an additional one-port FIFO is more efficient compared to utilizing three-port FIFO to store the samples. Furthermore, no multiplexer selector logic is required here.

4.4.2.2 Normalization

The design of the high-performance PE that can also compute the normalization of the lifting scheme is similar to the one found in the resource-aware PE. Three additional multiplexers are required to put this feature into the PE. **Fig**. 4.14 depicts the block diagram of the PE that has capability to compute the normalization. The same selector S5 is used to control these multiplexer, in order to maintain the compatibility with the previous PE. The only difference is that, because we have two multipliers here, the outputs of these multipliers are already synchronized. Thus, it is no longer necessary to buffer one of the multiplication results, i.e. no FIFO is required here.

4.5 The Need of Floating-Point

High-accuracy digital signal processing requires high-precision computations that cannot be satisfied by integer arithmetic computations. Floating-point numbers, which offer higher dynamic range, are used to overcome this problem. Floating-point representation does not only offer a better precision distribution for digital signal processing, but it also gives flexibility due to its dynamic range. Additionally, during the design concept of the hardware architecture of some signal processing blocks, one needs to verify that the given samples would not deliver any overflow in the fixed-point computation. If it is the case, the problem should be overcome by for example consuming extra bits or moving the decimal point. This issue is solved when the floating-point representation is used for the processing.

The major challenges on designing the floating-point arithmetic functions are the performance and the complexity. Among floating-point operations, addition and multiplication are the most used floating-point arithmetics in digital signal processing [**?**, **?**, **?**, **?**]. Numerous VLSI architectures for floating-point multiplier are already developed. References [**?**, **?**, **?**] detailed the architectures of a floating-point multiplier. In order to improve the performance, the architectures are pipelined with 2-level or 3-level depth, depending on the architecture and the data width selection.

Compared to the floating-point multiplication, the floating-point addition involves more steps such as exponent equalization, shifting, adding, and finding the first logic one for the normalization. Furthermore, floating-point addition has higher data dependencies that introduce a longer computation path. References [**?**, ?, **?**] discussed the architectures of a floating-point adder. Kowaleski *et al.* [?] described a 4-level floating-point adder

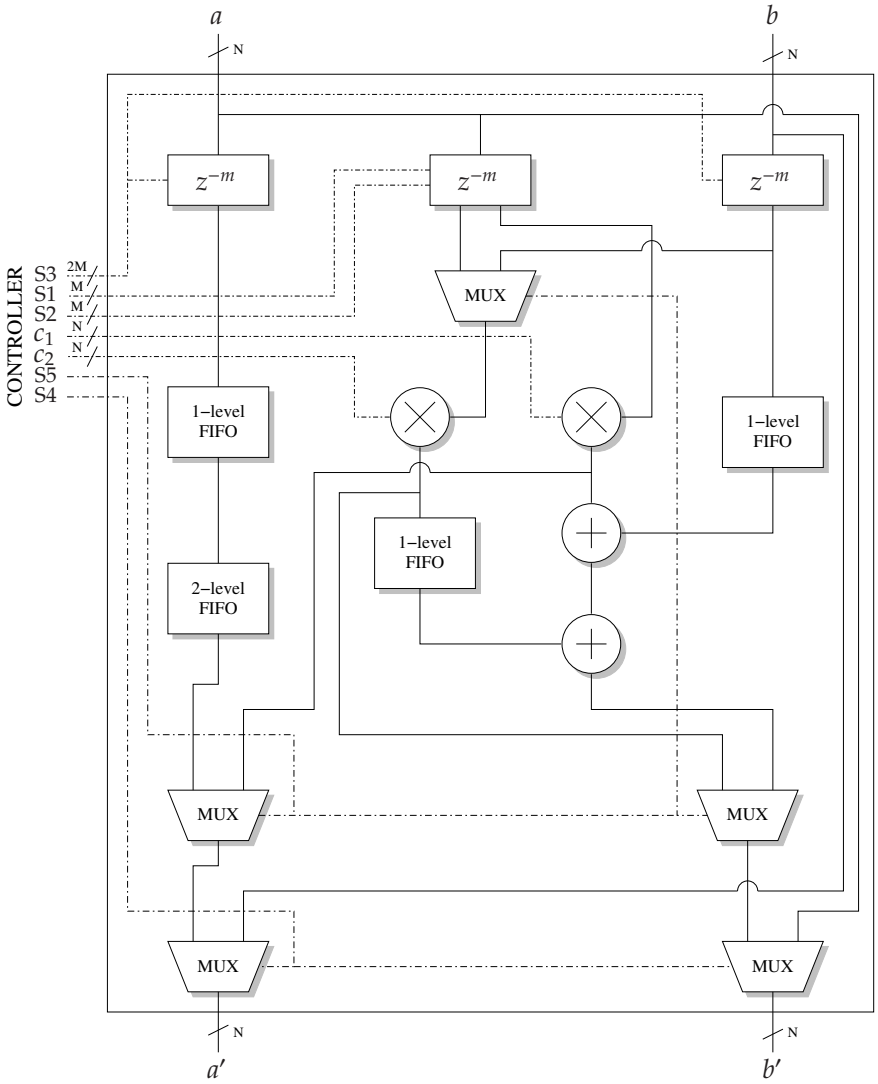


Fig. 4.14: Block diagram of the high-performance fixed-point PE with two multipliers and two adders and an additional capability to compute the normalization.

that can be operated at 433 MHz in a 0.35-μm technology and Beaumont-Smith *et al.* [?] detailed how to reduce the number of pipeline stages into three. Their architecture is synthesized using 0.5-μm technology and consumes 1.8 mm^2 chip area. Additionally, Pillai *et al.* [**?**] detailed the design of a floating-point multiply-and-add (FMAC) unit which is also common in digital signal processing.

Contrary to the multiplier, adder, and FMAC-unit, we have not found any reference regarding the works that has been done to design a floating-point adder that can accept three inputs. Due to data dependencies and the pipeline structure, two inputs will be added first and the temporary result will then be added to the third input in order to add three floating-point numbers. This introduces a longer pipeline structure with the normal approach (i.e. chaining two floating-point adders) and consumes more area.

DWT is a multiresolution signal analysis tool. In the traditional wavelet transforms,

only the low-pass signal is used on the next decomposition level, resulting octave frequency bands. In wavelet packets, both low-pass and high-pass signals are analyzed, resulting equally spaced frequency bands, as depicted in **Fig**. 4.7 and **Fig**. 4.8. The major issue in DWP is that the resulting high-pass components are much smaller than the low-pass components parts in normal circumstances. Thus performing multi-level DWP using integer arithmetics would make these high-pass components go to zero if it is not carefully performed.

Before we continue our discussion on the PEs that are based on floating-point arithmetic, we discuss the essential components for our PEs. From our previous discussion on the PEs that are based fixed-point arithmetics, we can see that two arithmetics operations are used, i.e. multiplication and addition. The floating-point implementations also follow the same principle. The following sections discuss the standard IEEE 745 floating-point format, followed by the architecture of 2-stage floating-point multiplier, and 4-stage and 3-stage floating-point adder with three inputs.

4.5.1 Standard IEEE 754 Format

The standard radix-2 binary floating-point format was issued by IEEE in 1985 [**?**]. This standard covers different types of floating-point formats (e.g. single, double), special coding representations (e.g. 0, $+\infty$, $-\infty$), rounding mechanisms, arithmetic operations, etc. The standard radix-2 binary floating-point representation can be written as:

$$(-1)^s \times f \times 2^e \tag{4.32}$$

with s as the sign bit, f as the mantissa or fraction, and e as the biased exponent (in binary excess representation).

4.5.2 Floating-Point Multiplier

There are many works carried out to design the floating-point multiplier [**?**, ?, ?, **?**, **?**]. Here we discuss our implementation apart from other existing architectures that extensively uses compound adder to push the performance to the higher speed. Our architecture of floating-point multiplier is relatively straightforward and we do not use a special adder to boost the performance. Instead, we leave the optimization of our floating-point multiplier to the underlying CAD tools.

As mentioned earlier, our floating-point multiplier is a two-stage multiplier. The floating-point multiplication follows the IEEE 754 standard [?]. Given are two floating-point numbers A and B:

$$A = (-1)^{S_a} \times M_a \times 2^{E_a} \tag{4.33}$$

$$B = (-1)^{S_b} \times M_b \times 2^{E_b} \tag{4.34}$$

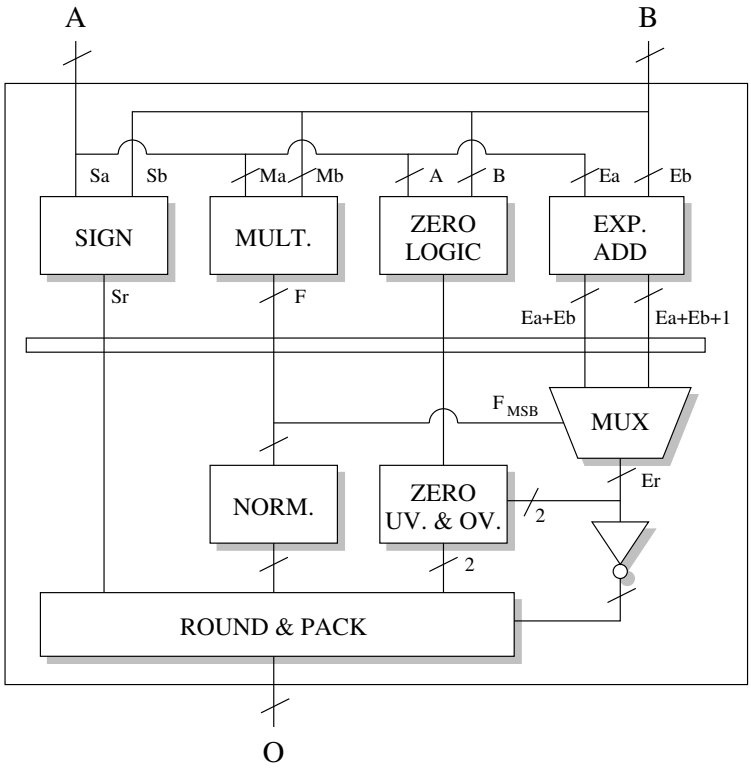


Fig. 4.15: Block diagram of the floating-point multiplier.

To multiply two floating-point numbers, we need to multiply the mantissas M_a and M_b and add the exponents E_a and E_b. Thus, we have:

$$O = (-1)^{S_a \oplus S_b} \times (M_a \cdot M_b) \times 2^{E_a + E_b} \tag{4.35}$$

We use two notations for multiplication, i.e. $\times$ and $\cdot$. They are basically the same and used here in order to give a better impression for the separation in treating the sign, mantissa, and also exponent during the hardware realization.

Several floating-point multiplier architectures are detailed in [?, ?]. Both architectures support only 32-bit single precision and 64-bit double precision formats. Our floating-point multiplier is a 2-level pipeline architecture and can be customized for other floating-point formats besides the standard single and double precision formats. Nonetheless, we also limit ourselves in the lower precision floating-point range, i.e. up to single precision. **Fig**. 4.15 depicts the block diagram of the floating-point multiplier.

4.5.2.1 Stage 1

At the first stage, the resulting sign is resolved by the *sign block* (i.e. $S_a \oplus S_b$, with $\oplus$ denotes an exclusive-or operation here). The *multiplier block* inserts the hidden bit to both mantissas M_a and M_b and performs the unsigned multiplication. As we stated earlier, we let the underlying CAD tools to optimize the operation of the unsigned multiplication. In other word, no compound adder is explicitly used here. By taking into account the amount of

Tab. 4.2: Truth table of possible resulting fraction after the unsigned multiplication.

Hidden Bit A	$M_{a_{MSB}}$	Hidden Bit B	$M_{b_{MSB}}$	F
1	0	1	0	0 1 0 0
1	0	1	1	0 1 1 0
1	1	1	0	0 1 1 0
1	1	1	1	1 0 0 1

the guard bits, the result of the multiplication will be truncated, resulting the fraction F. The amount of the guard bits that will be used by the *rounding block* is configurable. The *zero logic block* detects if one/both of the operands is/are zero. The *exponent add block* adds both exponents E_a and E_b. This block outputs two values to minimize the effort at stage 2. Because the exponents are in the biased-format, we can directly treat the operation as the normal unsigned addition. One guard bit is appended to the most significant bit of both exponents before the addition.

4.5.2.2 Stage 2

By examining the most significant bit (MSB) of the resulting fraction F, the *normalization block* normalizes the resulting fraction. The normalization is basically a shift-left by one operation. Additionally, this bit is also used to determine the exponent. The same technique is applied to select the correct exponent E_r. **Tab**. 4.2 summarizes the possible combinations that may occur on the resulting fraction after the unsigned integer multiplication. Note that this table serves as an illustration purpose only because here we consider the most significant bits of both operands only. Because the hidden bit is always one, except for the floating-point number (which is equal to zero), it is clear that normalization is in fact only either a shift-left by one operation or no shifting at all. By examining the MSF of the fraction F, we can also determine

- $F_{MSB} = 0 \Rightarrow E_r := E_a + E_b$
- $F_{MSB} = 1 \Rightarrow E_r := E_a + E_b + 1$

To convert the resulting exponent E_r back to the biased-format, we only need to invert the exponent (i.e. $\overline{E_{r[MSB-1:0]}}$). The *zero, underflow, and overflow block* checks if the result lies on the valid floating-point range.

- $Z = 1 \vee E_{r[MSB:MSB-1]} = 00 \Rightarrow ZeroFlag := 1$
- $E_{r[MSB:MSB-1]} = 11 \Rightarrow Overflow Flag := 1$

The *round and pack block* rounds the normalized fraction. Two rounding mechanisms are available: *rounding to zero* and *rounding to nearest*. The resulting sign, the rounded fraction, and the exponent are packed together. If the zero flag is set, the output will be cleared, and if the overflow flag is set, the output will be saturated.

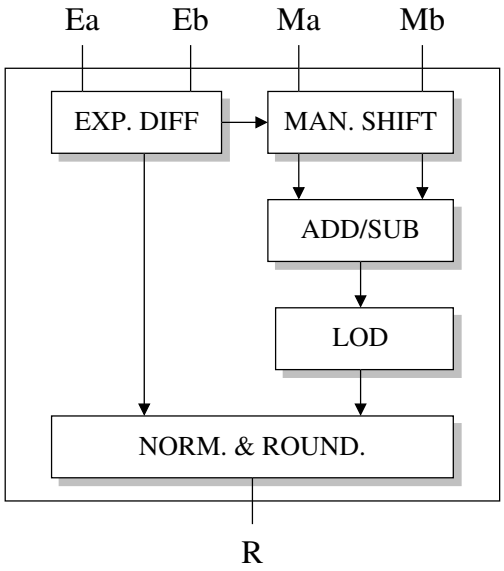


Fig. 4.16: Two-Input Floating-Point Adder with Leading-One Detector (LOD).

4.5.3 Floating-Point Addition Algorithm

Contrary to the floating-point multiplier, the floating-point adder requires more steps due to the algorithm complexity and the data dependency. As depicted in **Fig**. 4.16, to perform addition between two floating-point numbers, the following steps are performed:

1. Calculate the exponent difference.
2. Align the mantissa by shifting the mantissa with the lower exponent to the right.
3. Add/subtract both mantissas depending on the sign bits.
4. Perform the Leading-One Detection (LOD) to determine the location of the first logic one.
5. Normalize and round the result.

In order to decrease critical paths, Leading-One Prediction (LOP) was proposed in [**?**, ?, **?**, **?**] as a replacement of LOD, predicting the first occurrence of the logic one directly from the operands. **Fig**. 4.17 depicts the addition algorithm with LOP. The LOP works in parallel with the adder and it is based on the encoding tree, which examines both inputs from left to right. The LOP ignores the possible carry or borrow that might occur on the addition/subtraction result. Thus, it leads to one-bit inaccuracy, which will be corrected during the normalization step. This is why it is more popular with the name *inexact LOP*. With additional complexity, papers in [**?**, ?] detailed the design of the *exact LOP* that predicts correctly the position of the leading-one. Therefore, it minimizes the normalization effort.

References [?, ?, ?] discussed the architectures of a floating-point adder. Kowaleski in [?] described a 4-level floating-point adder that can be operated at 433 MHz in a

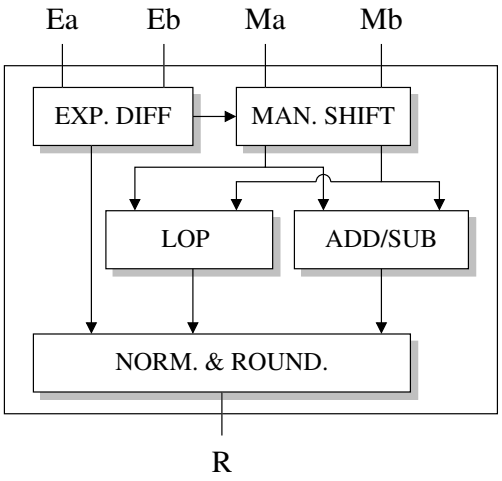


Fig. 4.17: Two-Input Floating-Point Adder with Leading-One Predictor (LOP).

0.35-μm process and Beaumont-Smith in [?] detailed how to reduce the number of pipeline stages into three. Their architecture is synthesized using 0.5-μm process and consumes 1.8 mm^2 chip area.

In order to add three floating-point numbers, one can use two floating-point adders with two inputs and add N-level register on the third input, which will lead to 6-stage pipeline for the architecture designed in [?], and at least an increase of logic counts with a factor of two. This introduces a longer pipeline structure with the traditional approach. With additional latency, the same architecture can be optimized by bypassing the first normalization step, which will save up one stage.

To the best of our knowledge, we have not found any reference regarding the research that has been carried out to design a floating-point adder with three inputs, to exploit the data dependencies in advance, in order to decrease the latency and the number of pipeline stages, and to minimize the amount of logics. To minimize the number of pipeline stages, we have developed a dedicated 4-stage 3-input floating-point adder [**?**, **?**]. The architecture of our proposed 4-stage floating-point adder with three inputs is unique in the way that the third input will be involved at the second stage. It is designed to suit with our resource-aware PE, but still maintains to reduce the latency. Furthermore, we also optimize our design in order to reduce the number of stages into three [**?**]. The 3-stage floating point adder will be used in our high-performance PE [**?**]. The next two sections discuss the architecture of both 3-input floating-point adder.

4.5.4 4-Stage Floating-Point Adder with Three Inputs

We propose a novel architecture to compute addition of three floating-point numbers. Our design is customizable. It supports the IEEE-754 single and double precision formats, and it can be easily configured to suit the application demand that requires different floating-point format besides the aforementioned formats. The block diagram of

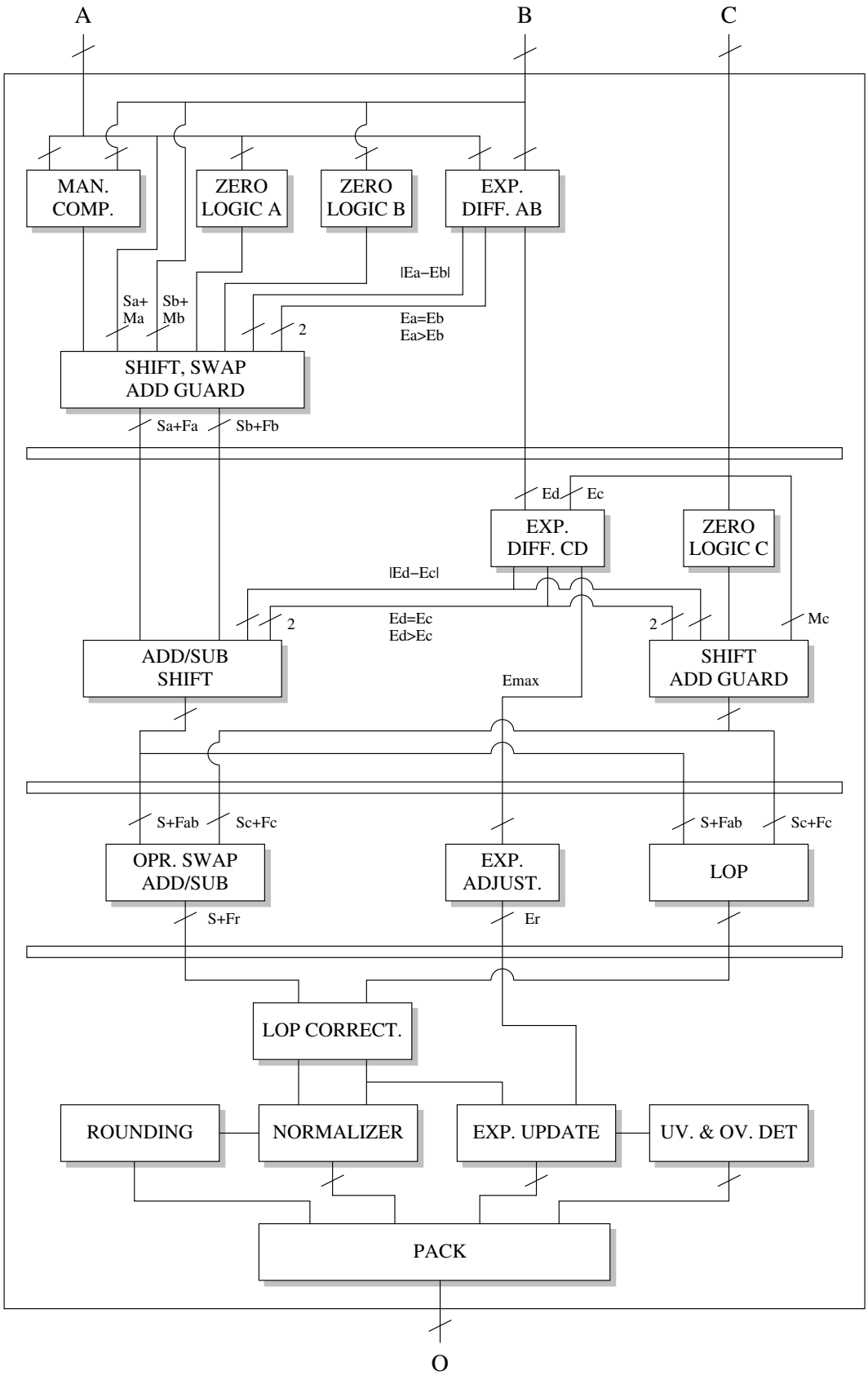


Fig. 4.18: The architecture of 4-stage 3-input floating-point adder.

the proposed architecture is shown in **Fig**. 4.18. The horizontal bars indicate the register pipelining. To maximize the performance, we have carefully examined the critical paths and utilized four pipeline stages. The exponent correction is divided into two stages to reduce the critical paths on the normalization stage, the LOP outputs two values to minimize the normalization effort, and the normalization stage is performed only once at the end of the stage to decrease the logic counts.

4.5.4.1 Stage 1

At the first stage, the two inputs A and B are unpacked. The *mantissa comparator block* compares both mantissas M_a and M_b and outputs one decision bit (i.e. $M_a \geq M_b$) which will be used by the *shift, swap, and add guard block*. The *zero logic block* detects if the corresponding input is zero. The *exponent difference AB block* compares both exponents E_a and E_b. This block outputs four different values. The first three values (the shift count $|E_a - E_b|$ and the comparator results $E_a = E_b$ and $E_a > E_b$) will be used by the *shift, swap, and add guard block*. The fourth value (the temporary dominant exponent $E_d = max(E_a, E_b)$) will be used by the *exponent difference CD block*.

Since the addition or subtraction is performed in a normal binary representation (i.e. unsigned number), it is necessary to sort/swap the operands (i.e. $M_a \leftrightarrow M_b$). We choose not to convert the numbers into signed, because LOP requires both inputs to be unsigned. Additionally, we do not need to convert the final result back to unsigned during the normalization step, which would require one extra stage before the normalization. As the name implies, the *shift, swap, and add guard block* aligns the mantissa M_a and M_b to have the same exponent degree by shifting the mantissa with the smaller exponent to the right. The hidden bit and the guard bits are appended to the most significant bit and the least significant bit of both mantissas respectively. The number of bits used as guard bits can be freely chosen. Based on the exponent difference, three different cases are examined here:

- $E_a = E_b$:
  Depending on the output of the *mantissa comparator block*, both mantissa M_a and M_b will be swapped directly (i.e. $M_a \leftrightarrow M_b$) without performing any shifting.

- $E_a > E_b$:
  The mantissa M_b will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$).

- $E_a < E_b$:
  The mantissa M_a will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$). Both mantissas will be swapped afterwards.

In all of the cases, if a zero number is detected, the corresponding mantissa(s) will be set to zero. The outputs of the *shift, swap, and add guard block* are the sorted and extracted

fractions F_a and F_b with their corresponding signs.

At this stage, the input C is not processed, thus two values that correspond to the prediction/update sample and the first multiplication result can be added first. The second multiplication result that comes on the next clock cycle will be unpacked at the second stage. The details will be clear when we discuss the architecture of the resource-aware PE that is based on floating-point arithmetic in **Sec**. 4.5.6.

4.5.4.2 Stage 2

At this stage, the fractions F_a and F_b are added/subtracted depending on the sign difference (i.e. $S_a \oplus S_b$), resulting in the fraction F_{ab}. If the exponent E_c is greater than E_d, the result will be shifted to the right. These steps are performed by the *add/sub and shift block* with the shift parameter determined by the *exponent difference CD block*. Three different cases as at the first stage are also examined here.

The *shift and add guard block* prepares the mantissa M_c. If the exponent E_c is less than E_d, M_c will be shifted instead. The hidden bit and the guard bits are appended to M_c, resulting in fraction F_c. Finally if the *zero logic block* detects a zero number, F_c will be set to zero.

4.5.4.3 Stage 3

At stage 3, the *operand swap and add/sub block* swaps the operands F_{ab} and F_c if necessary (note that both operands have the same exponent). Afterwards, it performs the addition or subtraction. To minimize the logic counts and the logic levels on this stage, we have utilized the inexact LOP. The *LOP block* works parallel with the *operand swap and add/sub block* to predict the first occurrence of the logic one directly from the operands. Taking into account that one-bit inaccuracy might occur on the prediction, the *LOP block* prepares two values at the output to minimize the critical paths on the normalization stage.

Because three addition/subtraction arithmetic operations are involved, the final result might have an increase of exponent by two. The *exponent adjustment block* prepares the dominant exponent by simply adding two to the largest exponent (i.e. $E_r = max(E_a, E_b, E_c)+ 2$).

4.5.4.4 Stage 4

Because the *LOP block* may deliver an error within one-bit degree, the error has to be corrected. The error can easily be detected by looking at the LOP-index bit of the resulting fraction F_r. This step is performed by the *LOP correction block*. Additionally, this block also performs the pre-normalization by shifting the resulting fraction F_r to the left with the shift amount determined by the inexact LOP value. We have examined that correcting the LOP result in real time (by adding with one) will increase the critical path on the

stage 4. This is why the *LOP block* on the stage 3 outputs two values. Therefore, we only need to choose the right value at the end.

Should the inexact LOP predicts the leading-one position falsely, the *normalizer block* corrects the pre-normalized fraction by shifting it to the left. Here we only need to perform one bit shifting. The rounding logic implements two rounding mechanisms: *rounding to zero* and *rounding to nearest*. Based on the corrected LOP value, the *exponent update block* updates the resulting exponent. The *underflow and overflow detector block* checks if the resulting exponent lies on the valid floating-point range. Finally, the sign, the normalized fraction, and the corrected exponent are packed together by the *pack block*.

4.5.5 3-Stage Floating-Point Adder with Three Inputs

In addition to the 4-stage floating-point adder, we also optimize the architecture by reducing the number of pipeline into three. From the previous discussion, it is clear that the first stage of the floating-point adder processes only two inputs whilst the third input is involved later on the second stage. It has advantage on our PE design that is based on time-sharing architecture, where two inputs are available at one clock cycle and the third input is provided at the next clock cycle.

In case of the design of high-performance PE, no time-sharing property is exploited and two multiplications are used. This means that all three inputs are available on the same clock cycle, when the reducing in latency time is what we are aiming. For this purpose, we optimize our 4-stage 3-input floating-point adder. Thus no additional buffer is required to latch the inputs.

Fig. 4.19 depicts our proposed 3-stage floating-point adder with three inputs. The major change takes place on the first two stages of the previous architecture with four stages. The *add/sub and shift block* is split into two in order to distribute the critical paths and the *exponent difference CD block* is optimized to be suit when it is directly chained with the output of the *exponent difference AB* at the first stage without introducing any pipeline register, but still maintaining the critical path low at this stage.

4.5.5.1 Stage 1

At the first stage, the inputs A, B, and C are unpacked. The *mantissa comparator block* compares both mantissas M_a and M_b and outputs one decision bit (i.e. $M_a \geq M_b$) which will be used by the *shift, swap, and add guard block*. The *zero logic block* detects if the corresponding input is zero. The *exponent difference AB block* compares both exponents E_a and E_b. This block outputs four different values. The first three values (the shift count $|E_a - E_b|$ and the comparator results $E_a = E_b$ and $E_a > E_b$) will be used by the *shift, swap, and add guard block* and the fourth value (the temporary dominant exponent $E_d = max(E_a, E_b)$) will be used by the *exponent difference CD block*.

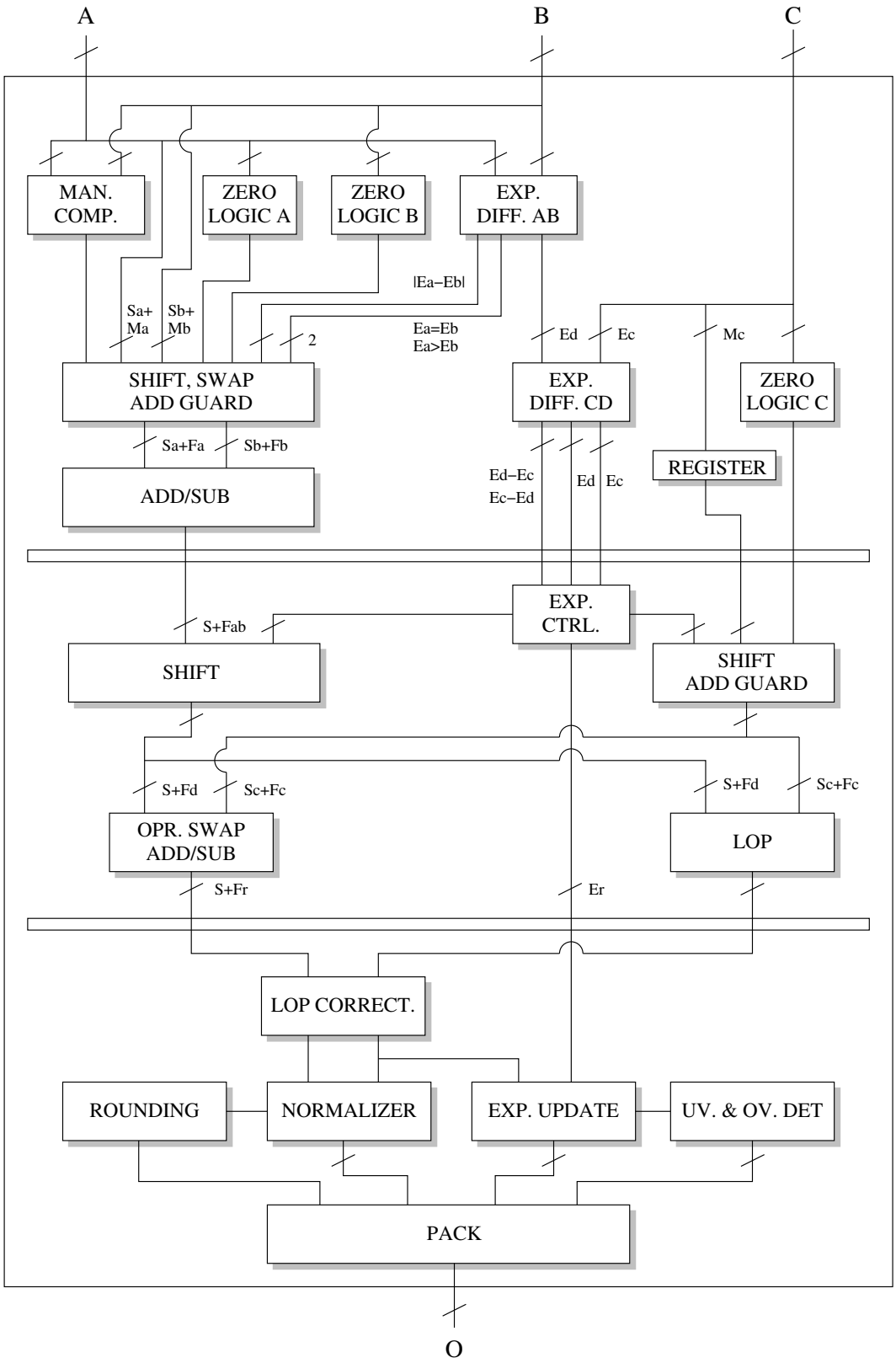


Fig. 4.19: The architecture of 3-stage 3-input floating-point adder.

As the name implies, the *shift, swap, and add guard block* aligns the mantissa M_a and M_b to have the same exponent degree by shifting the mantissa with the smaller exponent to the right. The hidden bit and the guard bits are appended on the most significant bit and the least significant bit of both mantissas respectively. The number of bits used as guard bits can be freely chosen. Based on the exponent difference, three different cases are examined here:

- $E_a = E_b$:
  Depending on the output of the *mantissa comparator block*, both mantissa M_a and M_b will be swapped directly (i.e. $M_a \leftrightarrow M_b$) without performing any shifting.
- $E_a > E_b$:
  The mantissa M_b will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$).
- $E_a < E_b$:
  The mantissa M_a will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$). Both mantissas will be swapped afterwards.

In all of the cases, if a zero number is detected, the corresponding mantissa(s) will be set to zero. The outputs of the *shift, swap, and add guard block* are the sorted and extracted fractions F_a and F_b with their corresponding signs. The *add/sub block* performs the addition or subtraction of the equalized fractions F_a and F_b depending on the sign difference (i.e. $S_a \oplus S_b$), resulting the fraction F_{ab}.

The *exponent difference CD block* is different compared to the *exponent difference AB block*. In order to reduce the logic levels, instead of calculating the absolute difference, this block outputs two differences (i.e. $E_d - E_c$ and $E_c - E_d$). The differences are computed and treated as unsigned numbers. One guard bit is appended to the most significant bits of both exponents before computing the differences. Both exponents E_d and E_c, and the mantissa M_c are latched.

4.5.5.2 Stage 2

By examining the most significant bit (MSB) of both exponent differences, the *exponent controller block* determines its outputs. Again, three different cases are examined here:

- $MSB_{E_d-E_c} = 0 \wedge MSB_{E_c-E_d} = 0 \Rightarrow E_d = E_c$
- $MSB_{E_d-E_c} = 1 \wedge MSB_{E_c-E_d} = 0 \Rightarrow E_d < E_c$
- $MSB_{E_d-E_c} = 0 \wedge MSB_{E_c-E_d} = 1 \Rightarrow E_d > E_c$

Because three addition/subtraction arithmetic operations are involved, the final result might have an increase of exponent by two. This block also prepares the dominant exponent by simply adding two to the largest exponent (i.e. $E_r = max(E_a, E_b, E_c) + 2$).

If the exponent E_c is greater than E_d, the temporary addition/subtraction result F_{ab} will be shifted to the right. These steps are performed by the *shift block* with the shift parameter determined by the *exponent controller block*.

The *shift and add guard block* prepares the mantissa M_c. If the exponent E_c is less than E_d, M_c will be shifted instead. The hidden bit and the guard bits are appended to M_c, resulting in fraction F_c. Finally if the *zero logic block* detects a zero number, F_c will be set to zero.

The *operand swap and add/sub block* swaps the operands F_d and F_c if necessary (note that both operands have the same exponent). Afterwards, it performs the addition or subtraction. To minimize the logic counts and the logic levels on this stage, we have utilized the inexact LOP. The *LOP block* works parallel with the *operand swap and add/sub block* to predict the first occurrence of the logic one directly from the operands. Taking into account that one-bit inaccuracy might occur on the prediction, the *LOP block* prepares two values at the output to minimize the critical paths on the normalization stage.

4.5.5.3 Stage 3

Because the *LOP block* may deliver an error within one-bit degree, the error has to be corrected. The error can easily be detected by looking at the LOP-index bit of the resulting fraction F_r. This step is performed by the *LOP correction block*. Additionally, this block also performs the pre-normalization by shifting the resulting fraction F_r to the left with the shift amount determined by the inexact LOP value. We have examined that correcting the LOP result in real time (by adding with one) will increase the critical path on the third stage. This is why the *LOP block* on the second stage outputs two values. Therefore, we only need to choose the right value at the end.

In case of the inexact LOP predicts the leading-one position falsely, the *normalizer block* corrects the pre-normalized fraction by shifting it to the left. Here we only need to perform one bit shifting. The rounding logic implements two rounding mechanisms: *rounding to zero* and *rounding to nearest*. Based on the corrected LOP value, the *exponent update block* updates the resulting exponent. The *underflow and overflow detector block* checks if the resulting exponent lays on the valid floating-point range. Finally, the sign, the normalized fraction, and the corrected exponent are packed together by the *pack block*.

4.5.6 Resource-aware Floating-Point PE

After we discuss the required components to implement the floating-point arithmetics, we continue our discussion on the design of the PEs that use these features [**?**, **?**, **?**]. The main concept of the PE that carries floating-point arithmetic inside of it is the same as the one found in the fixed-point implementation. Nevertheless, because the floating-point multiplier requires two clock cycles to do its job, and dedicated three-input floating-point adder are in the use, some changes are expected.

4.5.6.1 Processing Element

The first architecture that involves the floating-point arithmetic inside the PE is designed to minimize the hardware cost. It is well known that the multiplier consumes more logic compared to the adder in the fixed-point implementation. Likewise, in order to reduce the number of logic, we exercise only one floating-point multiplier that is time-shared to compute both prediction/update factors.

Fig. 4.20 depicts the PE that utilizes one floating-point multiplier. As previously stated, the multiplier is time-shared. It means that the results of the multiplier come alternately one clock cycle after another. Taking this into account, we develop a dedicated floating-point adder that can accept three inputs, where two inputs are processed first and the third input is fed one clock cycle later. The detail of this architecture is found in **Sec**. 4.5.4. Using this 4-stage 3-input floating-point adder, we optimize the PE by reducing its latency time.

The same as the fixed-point realization, the PE is a pipeline-based architecture. The PE has two selectors S1 and S2 to choose the prediction or the update samples. Two constants c_1 and c_2 are also defined outside the PE. Selector S3 controls the prediction or update that requires future samples and selector S4 is a bypass selector. Two unit delays are implemented on both input ports a and b. In order to reduce the number of registers needed for the unit delay, the unit delay on port a has two input selectors (i.e. S3 and the multiplexer output) and two outputs. Two 2-level FIFOs on both input samples are implemented to compensate the multiplier delay. One 4-level FIFO is implemented to compensate the delay introduced by the adder.

It is now come to play the dedicated architecture of the 4-stage floating-point adder. One might notice that since the multiplier is time-shared, it means that it requires 2+1 clock cycles to perform two multiplications. In addition to that, 4-level FIFO is used by the adder. In total, 7-level FIFO on the left path is needed to synchronize its output with the right path. In contrary, we only need FIFO with 2+4 depths to make both output synchronized. This saving is done by feeding the first multiplication result (i.e. $c_1 m_1$) and the reference sample (e.g. b) at first at port A and B of the adder, and the second multiplication result (i.e. $c_2 m_2$) at port C one clock cycle later.

This one clock cycle saving might not be seen as an important issue, but when we consider that the PE will be chained to another PE, and the wavelet processor might contain for example eight PE, the delay will add up.

4.5.6.2 Normalizer

Same technique is applied here in order to support the normalization for the PEs that are located at the top and at the bottom chain. **Fig**. 4.21 depicts the case. Three additional multiplexers are needed to add the normalization factor unit into the PE. By enabling S5 and setting S1 and S3 to zero, two inputs of the multiplexer before the multiplier corre-

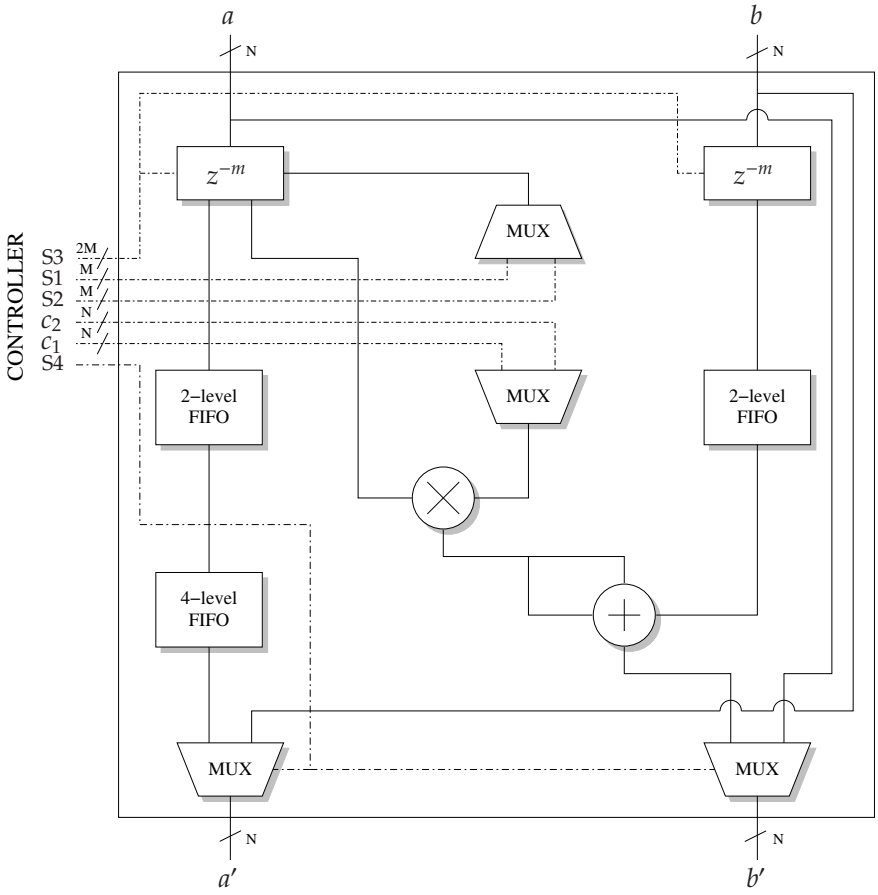


Fig. 4.20: Block diagram of the resource-aware floating-point PE with one multiplier and 4-stage adder.

spond to the actual samples a and b (with the normalization factors $K = c_1$ and $1/K = c_2$). The first multiplication product passes through the multiplexer and the 1-level FIFO resulting in $a' = Ka$ (left side). The second multiplication product passes through the multiplexer resulting $b' = b/K$ (right side). The 4-level FIFO is split into 3-level and 1-level FIFOs, with the latter used to make the both outputs synchronized.

4.5.7 High-Performance Floating-Point PE

Similar approach is also taken in order to provide floating-point arithmetic core into the high-performance PE, that is the PE that exercises two floating-point multipliers. Since we use two multipliers, both results of the multiplications are delivered at the same time in every clock cycle. Together with the third sample that comes from path b, we can either use the same 4-stage floating-point adder architecture by adding one extra delay unit (i.e. 3-level FIFO instead of 2-level FIFO) in the path b, or we can use the optimized 3-stage 3-input floating-point adder described in **Sec**. 4.5.5. The later contributes one clock cycle less computation latency. That is why it is chosen in the design and implementation of this PE [?].

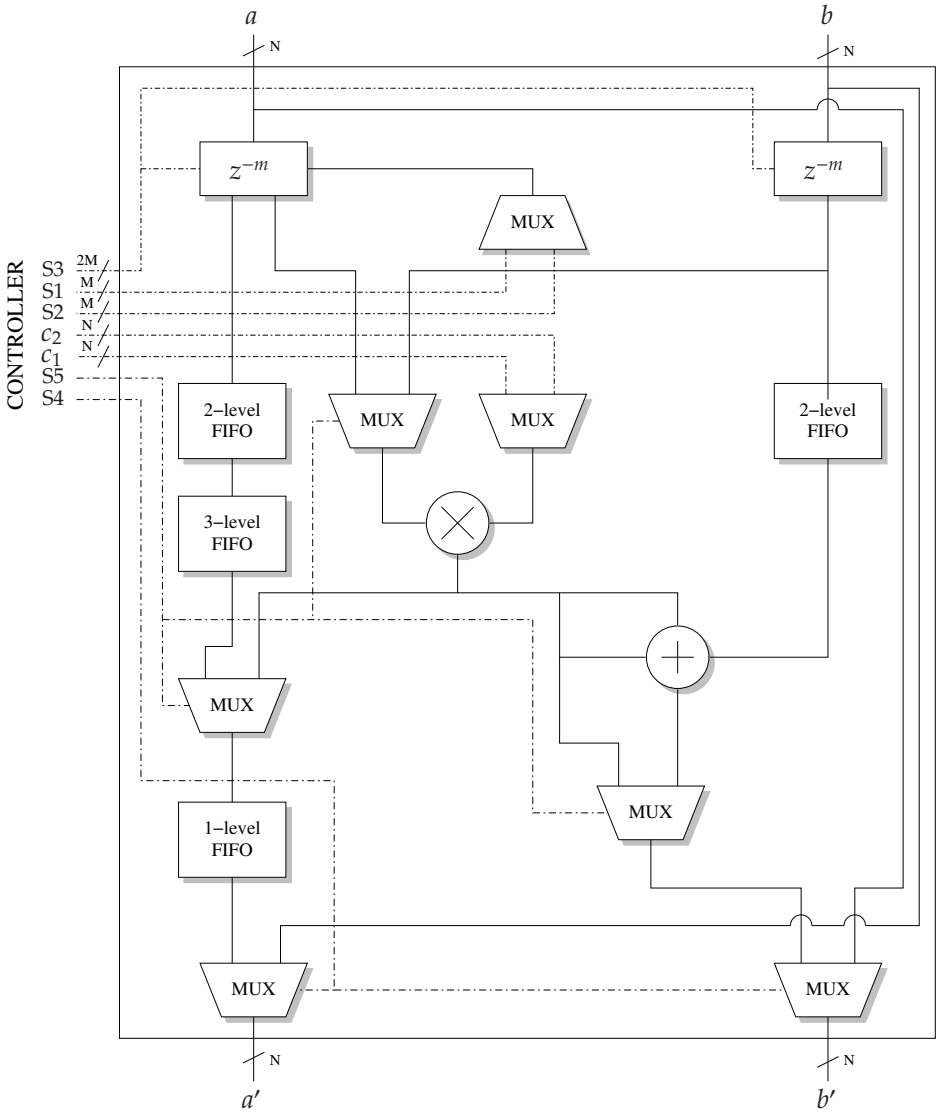


Fig. 4.21: Block diagram of the resource-aware floating-point PE which is located on the top and on the bottom of wavelet processor.

4.5.7.1 Processing Element

Fig. 4.22 shows the block diagram of the floating-point-based PE that uses two multipliers. The architecture is similar to the corresponding fixed-point implementation. The differences can be spotted directly that the floating-point-based PE requires more FIFOs to synchronize both outputs and the adder uses the our 3-stage and 3-input floating-point adder detailed in **Sec**. 4.5.5. Two-level FIFOs are required to compensate the delay introduced by the floating-point multiplier and one 3-level FIFO is used to compensate the delay introduced by the adder. The interfaces are maintained to have exactly the same interfaces as we can find in the three previous implementations. Thus, replacing one type of PE with another type of PE will not be an issue. One final remark, although the interfaces of the PEs are exactly the same, we cannot mix one type of PE with another type of PE in

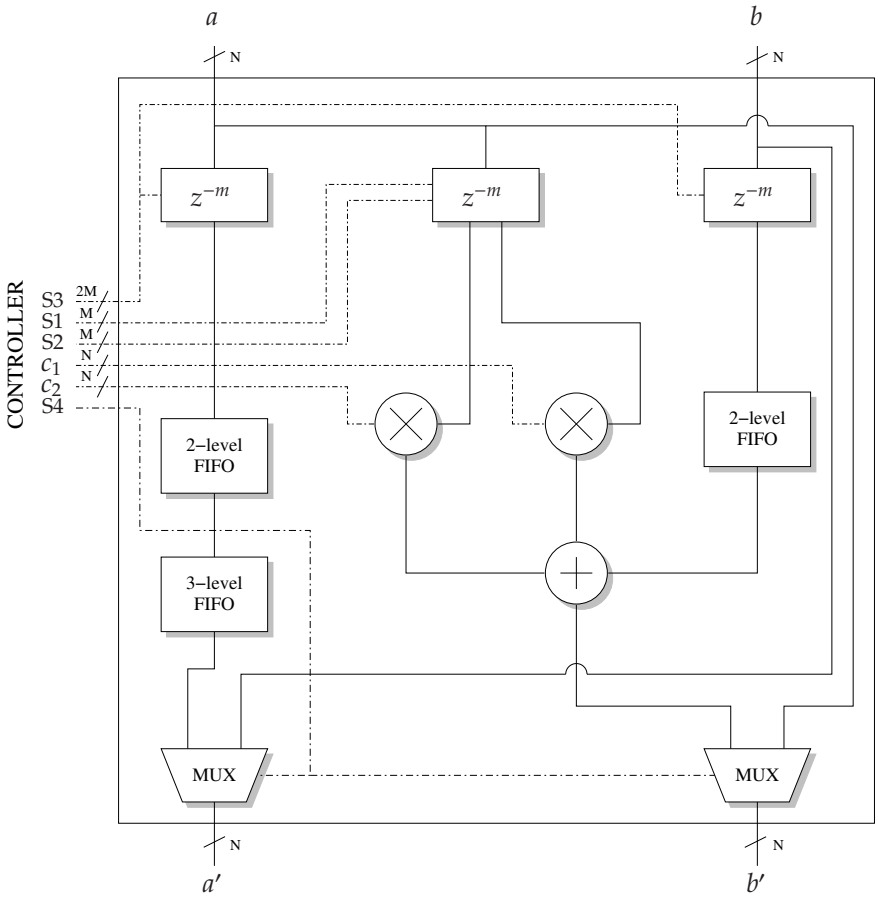


Fig. 4.22: Block diagram of the high-performance floating-point PE with two multipliers and 3-stage adder.

the realization. The reasons are the fixed-point PEs have different numbering representation compared to the floating-point ones, and the resource-aware PEs can process data every two clock cycles whereas the high-performance PEs is able to process data every clock cycle.

4.5.7.2 Normalizer

The floating-point architecture of the PE that supports normalizer can be seen in **Fig**. 4.23. The same concept as the design of the normalization on the other PEs is taken here; three additional multiplexers are needed to implement this feature. By enabling S5 and setting S1 and S3 to zero, the normalization can be activated in this PE.

4.6 Context Switch for PE

As we can notice from the drawing of all PEs, the data flows, which correspond to the even samples $s^{(i)}$ and odd samples $d^{(i)}$, are drawn on the top and the bottom sides of the

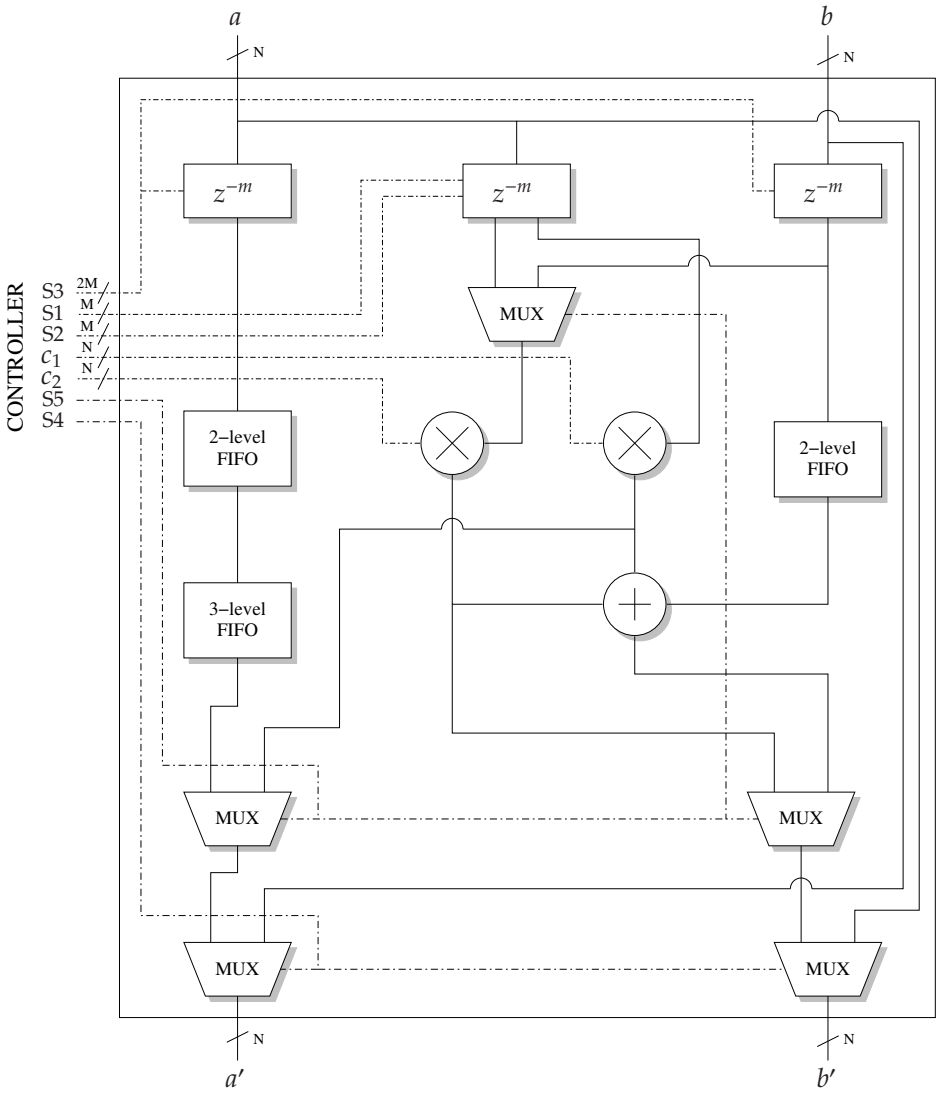


Fig. 4.23: Block diagram of the high-performance floating-point PE which is located on the top and on the bottom of wavelet processor.

block diagram while the configuration related parameters that control the functionalities of the PEs are drawn on the left side of the diagram. These configurations related parameters can be divided into two categories. The first category belongs to the coefficients of the atom function and the second category belongs to the power factors of the Laurent polynomial and the normalization. These configurations provide all necessary parameters to configure the atom function.

We have separated the configuration dependent parameters from the PE to cope with various lifting-based discrete wavelet decompositions and also discrete wavelet reconstructions. Apart from that, the PE is also designed to be simple. Thus, no finite state machine is required to control the PE. This also means that it is not necessary to control the PE from the configuration interface, which also simplifies the design of the configuration block. This leads us to a passive configuration storage mechanism, which is more

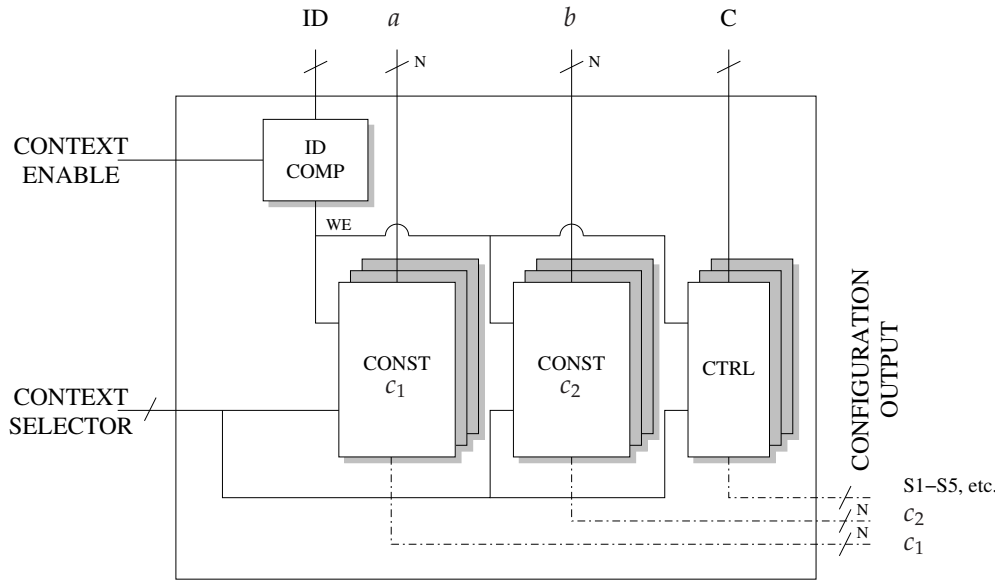


Fig. 4.24: Context switch for the PE.

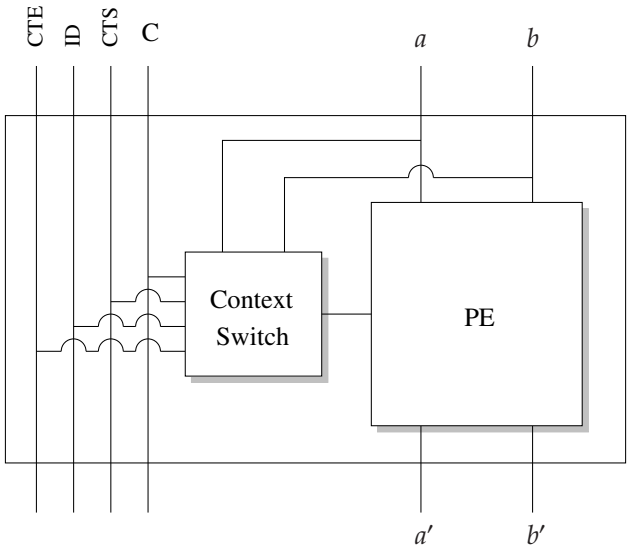


Fig. 4.25: Functional unit that consists of a PE and its context switch.

popular with the name context switch.

To support different classes of wavelet filters that require different types of configurations, we have implemented a multi-context configuration on each PE as depicted in **Fig**. 4.24. Note that although it is separately drawn, the context switch is actually bounded to the PE. In this way, the PE can be addressed using its ID. Each PE is assigned with a row index as a unique ID for the configuration. The first category of the configuration, that is the two coefficients that correspond to the Laurent polynomial or the normalization factor, use the data paths to save the wiring cost whereas the multiplexers configuration which belongs to the second category requires additional controller path. Context switch is implemented as a memory module where the address is controlled by

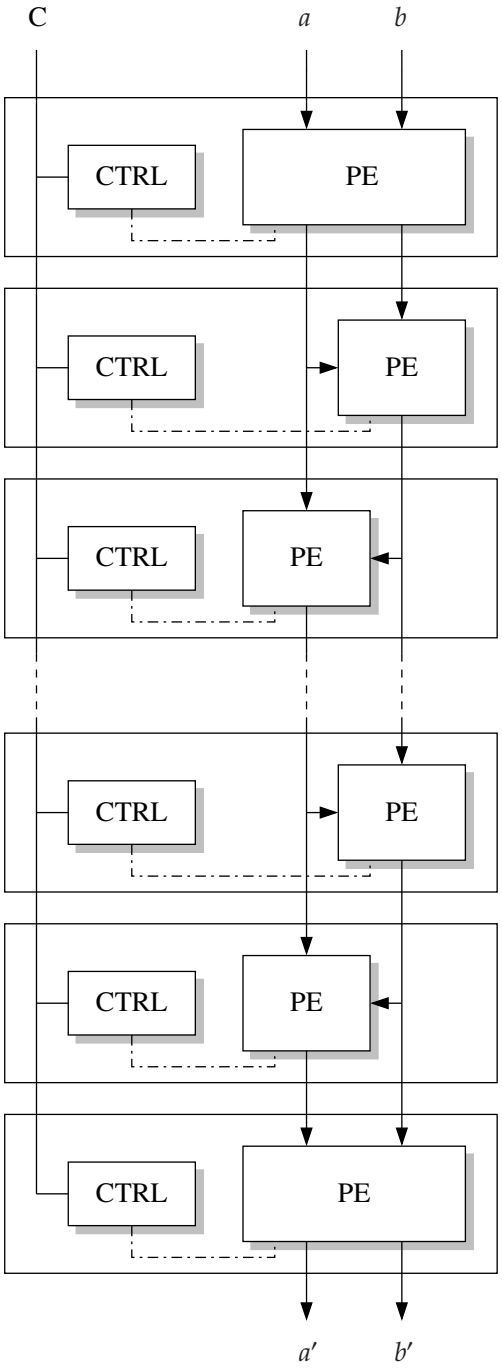


Fig. 4.26: Cross-chained PEs.

the context selector and the write enable signal is controlled by the output comparator and the context configuration enable.

A better representation of the block diagram of the PE and its context switch is depicted in **Fig**. 4.25. CTE corresponds to the context enable signal and CTS corresponds to the context select. This figure also shows that the PE and its context switch can easily be chained with another PE in an easy way. These chained PEs arrangement can easily be configure using one common configuration interface (i.e. from the PE that is located at

the top of the stack), as depicted in **Fig**. 4.26.

The active configuration can easily be selected by using this context-based controller to cope with various wavelet filters. One benefit of having a multi-context configuration is that the proposed wavelet processor can be configured to perform the corresponding discrete wavelet decomposition and discrete wavelet reconstruction in a very simple manner. Additionally, by using the context-based configuration, the wavelet transforms that exercise longer wavelet filters can simply be broken into smaller lifting steps. The configuration of each group of the lifting steps will be stored in the context memory and will be used to compute the transform.

4.7 Common Components

Previous sections detail the architectures of the PE, starting from the resource-aware fixed-point PE, up to the challenge in providing floating-point arithmetic core inside the PE to deliver higher flexibility and also higher performance. The PE alone does not do much except performing atom function of a predictor or an updater. This section details the rest of the components that build up the wavelet processor.

Taking into account that the predictions and the updates occur alternately, the outputs of a PE will be cross-linked with the input of the next PE. Due to the nature of lifting steps and also the architecture of the PE, the prediction and the update are computed in-place. It means that basically, it is not necessary to save the result or the temporary result into a different memory. One simple implementation of the proposed wavelet processor would consist of one PE. By configuring each context with the corresponding lifting step, the discrete wavelet decomposition or reconstruction could be computed with this simple implementation. Although it is possible to use only one PE, a typical wavelet processor will have N chained PEs configuration to boost the performance and to minimize memory access. These PEs are cross-linked to deliver alternating prediction and update atom functions. Additionally, the PEs that are placed at the top and at the bottom of the chain are configured to support the normalization unit whereas the PEs located in the middle do not need such feature.

Wavelet transform is a multiresolution signal processing tool. To achieve the required results, the signal needs to be transformed iteratively. In case of a decomposition of a DWT, only the low-pass component of the signal is taken into account as an input for the next transform. As a pair of low-pass and high-pass wavelet filter is used to compute the transform, the size of the signal decreases by two after each transformation level in this case. In contrary, a decomposition of a DWP uses both low-pass and high-pass components of the signal in order to achieve equally spaced frequency bands after each transformation level. The total size of the signal on DWP remains the same and the amount of the processed data will slightly increase. It is due to the fact that low-pass and high-pass components are treated independently during the computation and for each part

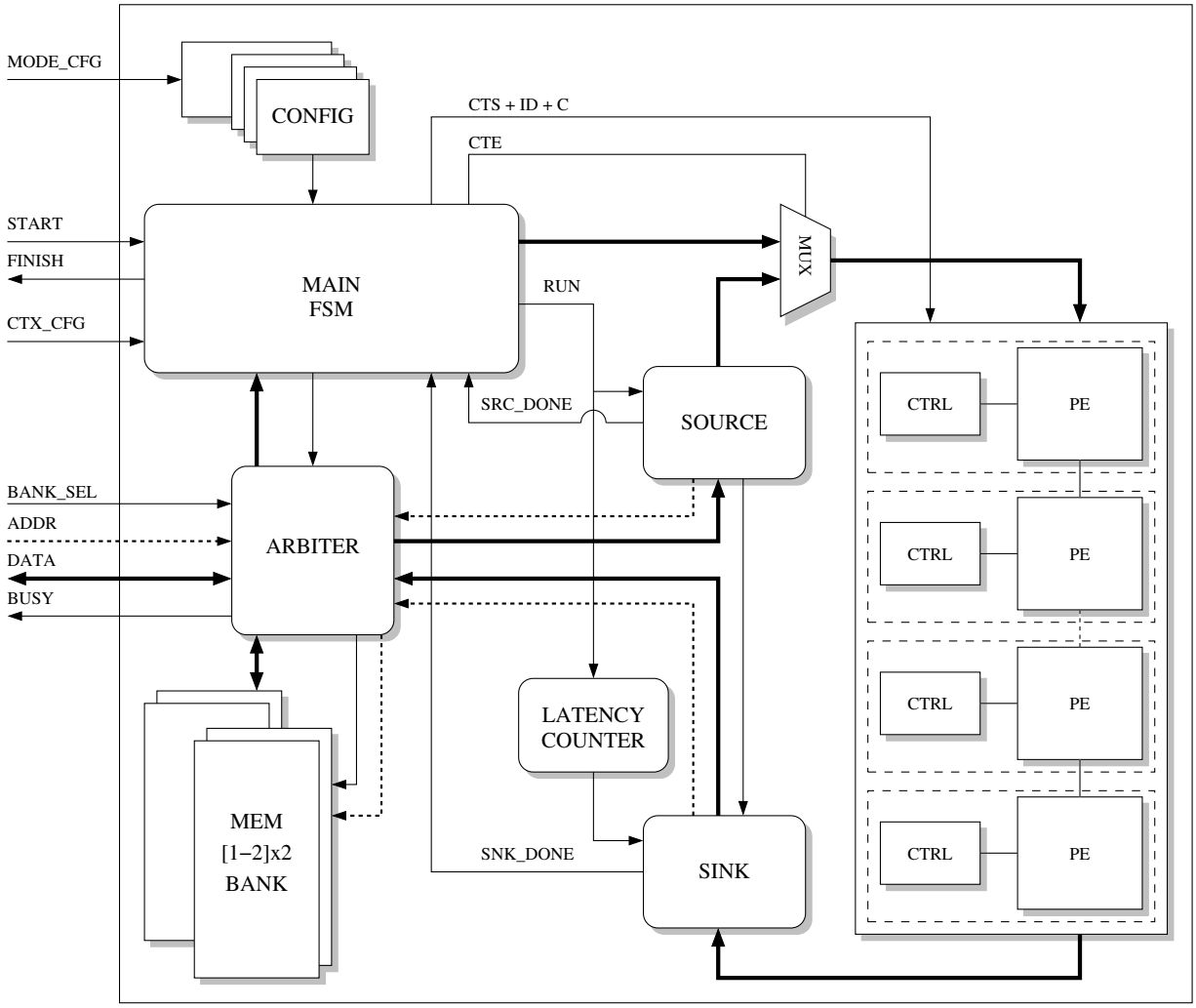


Fig. 4.27: The Proposed Wavelet Processor.

of the signal, a signal extension, which will be detailed later on, is required to compute the transform on the boundary regions. This is true for all wavelet filters except Haar wavelet.

Fig. 4.27 depicts the block diagram of the processor along with the PEs and their configuration controller [?, **?**, ?]. The PEs that are located on the top and on the bottom of the wavelet processor have an extra capability to perform the normalization while the PEs in the middle do not require this function. The rest of the section details the components depicted in the figure.

4.7.1 Main FSM

The main finite state machine controls the wavelet processor. **Fig**. 4.28 shows the state diagram of the FSM. When the transform is initiated, the FSM reads the necessary configurations, such as the transformation level, forward/inverse mode (that is, either wavelet

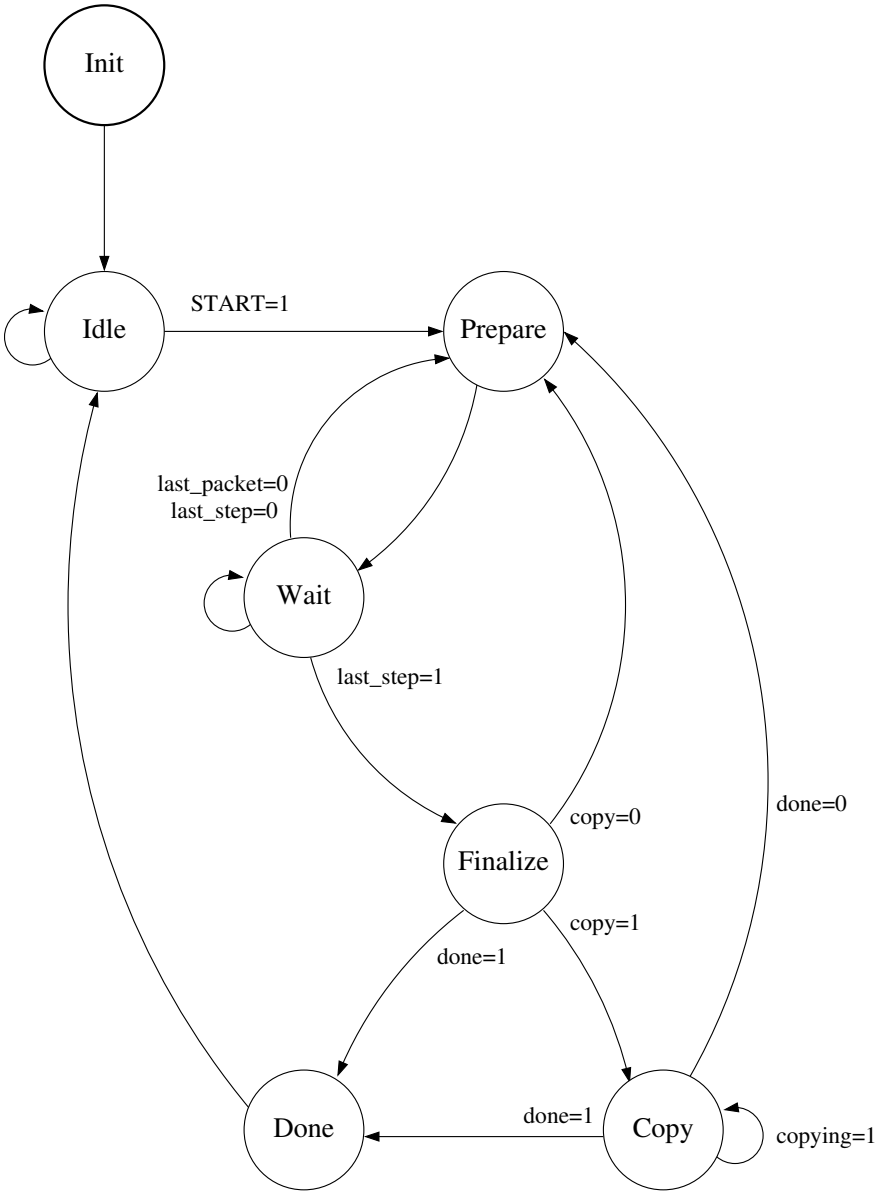


Fig. 4.28: Main FSM that controls the wavelet processor.

decomposition or reconstruction will be performed), transform/packet mode, used contexts, etc. from the *config* block. This configuration, as detailed later, is divided into two categories. The first category is related to the functionalities of the processor and the second one is related to the lifting configuration.

The main FSM prepares the source and the sink addresses where the data will be read and stored, and also the length of the data needed to be processed. We exploit the periodicity extension to cope with the boundaries issue in order to compute the transform on those regions. This implies that source address does not always start on the top of the page. Address masking techniques are applied here to localize the page. The FSM takes care of the possibility of having a longer wavelet transform that has to be split into several lifting steps on the target PEs. The FSM allows multi-level discrete wavelet

decomposition and reconstruction for both DWT and DWP to take place by means of iteration process.

The brief of task definitions on each state are given below. The detail of how the transforms are performed by the processor is detailed later.

Init This state is the first state when the processor is released from the reset state. It initializes the registers and all necessary configurations.

Idle This is the main state. It waits until the transform is initiated. During the idle state, the processor does not take the ownership of the address and data lines. Thus, the data can be prepared by the external interface. Additionally, during this state the context switch configuration for the PEs and also the processor related configuration can be set up. Once the transform is initiated, the configurations regarding the type of the transform (i.e. DWT or DWP), the mode of the transform (i.e. decomposition or reconstruction), the transform level, the number of samples, address masks, etc. are loaded from the configuration module to its internal registers.

Prepare Once the context switch configuration is set, the processor is configured, the samples are prepared, and the transform is started, the FSM configures its parameters. It prepares the starting address for the source and for the sink, and also the length of the samples involved, from the values given by its internal registers. Finally, the FSM tells the source and the sink to start their process by triggering the *run* signal.

Wait The FSM waits until the source and/or sink block finish their assigned task. The FSM has capability to perform discrete wavelet decompositions and reconstructions that use longer wavelet filters, which contribute to longer lifting steps. If the number of the lifting steps is greater than the available PEs, we can split these lifting steps into smaller steps. The FSM will check whether the transform currently performed uses this feature. If it is the case, the FSM will switch the bank and perform the next group of the lifting steps.

Finalize The prepare and wait states accomplish the one level decomposition or one level reconstruction. If the processor is configured to perform multilevel transform, we need to iterate the transform process. In case of a DWT decomposition for example, the samples involved for the next transform is halved.

Copy Because of the bank switching activity, the high-pass components might need to be copied to the other bank. If it is the case, the FSM prepares the source and the sink to copy these samples to the other bank.

Done Once the transform is finished, the FSM goes to this state and triggers the *finish* signal to inform the external interface that the transform is completed.

4.7.2 Config

The config block contains the configuration of the wavelet transform. **Tab**. 4.3 summarizes the parameters stored in configuration block. Two different configuration categories are managed by this block. The functionality part manages:

- Selecting the type of the transform that will be performed. The processor supports two different wavelet transforms, DWT and DWP. Contrary to the DWT that is supported by default in our architecture, the feature to enable the DWP can be selected during the design time. Nevertheless, the wavelet processor that is designed to support DWP has also the capability to perform the traditional DWT.

- Selecting the transform mode: discrete wavelet decomposition or reconstruction.

- The amount of memory, that is the number of samples, that will be involved during the transform. Note that the processor can perform the transform on an arbitrary size of the sample. For an example, the value 0 indicates that the transform will be performed on the whole memory. The value 1 will make the transform processes the half of the memory and so on.

- Number of levels the transform will compute. This is effective to perform multilevel transform on a 1D signal. In contrary, for a 2D or higher dimension, the number of levels should always be set to 1.

The lifting part stores the configuration of the contexts used during the transform. It holds an important key to support wavelet transforms that use longer wavelet filters. If the number of lifting steps of the wavelet filters used for the transforms are larger than the available PEs, these lifting steps have to be split into several smaller steps that can be fit into the available PEs. The configuration of each lifting itself is stored on the context configuration of the PEs. This block stores only the corresponding context IDs that will be used. Thus, by selecting the right ID one after another, the wavelet transform with longer lifting steps can be performed. Basically, it tells us which context should be used for the corresponding lifting step.

Besides storing the context IDs, it also holds the read and write offset addresses to start the transform and also the latency value for each lifting. It is important to note that in order to compute the wavelet transform, except for Haar wavelet filter, past and future samples are required. This becomes an issue when the transform on the signal boundary is performed. To cope with this boundary issue, the periodicity extension is used to locate these samples. These offsets hold the information of the corresponding starting sample for this periodicity extension.

Tab. 4.3: Description of the parameters stored in configuration block.

Name	Desciption
SIZE_OFS	The size of the memory used for the transform. It is a zero-based value. Value 0 indicates the whole memory, 1 indicates the half of the memory, and so on.
DWP_OFS	It is used for DWP. It indicates the starting offset of the DWP. Value 0 indicates the samples are treated as one low-pass components, 1 indicates the input samples have two components, etc.
LEVEL	The number of decompositions or reconstructions to be performed. Value 0 indicates one-level decomposition/reconstruction, 1 indicates two-level decomposition/reconstruction, etc.
SWAP	For some wavelet filters, prediction might come before update. This parameter informs the processor to swap the even and odd samples.
STEPS	The number of the group lifting steps that will be processed. It is useful for the wavelet filters that occupy more than the available PEs.
LS_CONTEXT[]	This is a register bank. It points out which context will be used to compute the transform.
LS_OFSREAD[]	Because most of wavelet filters require previous samples. It will become an issue to compute the transform at the boundary region. As a solution, periodicity extension is used to overcome this problem. This offset tells the processor at which location the first sample from the periodicity extension resides.
LS_OFSWRITE[]	Similar to the read offset, it tells the processor at which location the first resulting sample should be stored.
LS_LATENCY[]	Because the PE is a pipeline-based architecture, the computation latency is expected. This latency varies on every lifting. It indicates the latency between the first sample that is pushed to the PE and the valid computed samples that is outputted by the bottom PE.

4.7.3 Memory

Originally the organization of the memory required to support our wavelet processor is arranged as 1×2 banks. This configuration tells us that two banks, called primary bank and shadow bank, exist and can be used to load the samples and store the transform results. We use two banks configuration not only to ease the implementation, but also to support in-place group lifting schemes computation for the wavelet filters that cannot be fit into the available PEs directly.

Additionally, this memory bank organization simplifies the data storing and data management for further signal processing. This can be explained as follows. Let us assume that we use only one bank to load the samples and also to store the transform results. When the wavelet decomposition is performed, its results, i.e. low-pass and high-pass components, have to be stored in a sequence manner in the memory. It is not

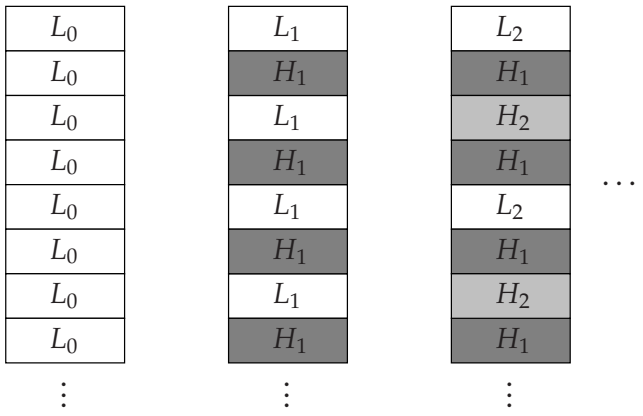


Fig. 4.29: The content of the memory using one bank configuration only.

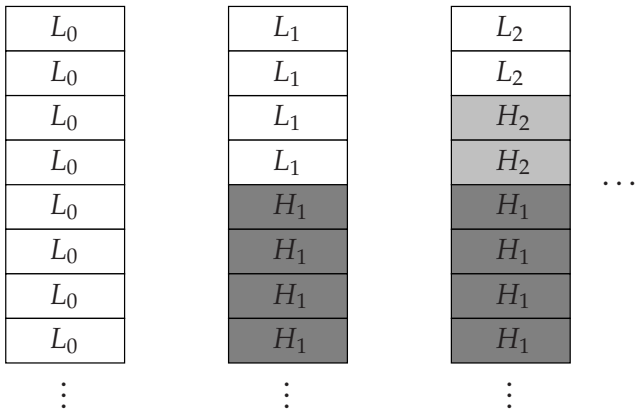


Fig. 4.30: The content of the memory using two banks configuration.

possible to split both components into two different memory pages due to the fact that the second page still contains the original samples that have not been loaded and processed. This will become more complex when multilevel transform is performed. The low-pass decomposition results will be stored every $2N$ interval and the rests will be occupied by the high-pass components. **Fig**. 4.29 depicts the case. Note that here we use different notations to represent the multiresolution signal analysis space. L_0 represents the original space, L_1 represents the lower resolution of the L_0, H_1 represents the details extracted from L_0, etc. The left part of the figure shows the original samples, the middle part is the result after one-level decomposition, and the last part is the result after two-level decomposition. In contrary, by using two banks approach, we have a better samples organization. The resulting low-pass and high-pass components can be stored in two different pages every transform level. **Fig**. 4.30 illustrates the case of the content of the memory by using two banks configuration.

The memory write and read accesses are exclusive, which means that writing to the memory will write to the primary bank and reading from the memory will read from its shadow. This state is switchable automatically, controlled by the *FSM*. When the transform takes place, the *FSM* grants the memory access to the *source* and *sink* blocks. Writing

to or reading from this bank is forbidden and it will generate an error as an indication of a busy signal. This mechanism simplifies the memory access, the design of the *FSM*, and it is straightforward, which means that the primary bank always acts as the source to load the samples and the shadow bank is always used to store the transform results. If we examine **Fig**. 4.30 carefully, the use of two banks also has disadvantages. One of them is referring to the previous high-pass components (in this case H_1) that have to be copied to another bank. That is why the *FSM* has additional state to perform this operation.

Because the wavelet processor is not only targeted for one specific application, we also improve our memory architecture in order to reduce the latency time. This processing latency is contributed mainly during the data preparation, waiting for the resulting transforms, and finally storing the resulting transforms elsewhere. In case of a 2D or higher dimension transforms, the major bottleneck happens in preparing the samples for the transform. As an example, in a 2D image signal, 1D transform is performed for every row image, followed by the 1D transform for every column image, resulting LL, LH, HL, HH components. The LL component will be processed with the same technique to get the multiresolution signal analysis space representation of that signal. The first column image transform can only be started after all the row image samples are processed and the whole samples need to be processed in order to get the LL component. As one can notice, because the wavelet processor utilizes pipeline mechanism, loading the samples and saving the resulting transforms consume more time than the computing the transform itself. In order to reduce the data preparation time, we also design a memory with 2×2 banks configuration, where the main banks are called bank 0 and bank 1, and each main bank has two banks, logically called primary and shadow banks. By introducing two main banks, when the transform is initiated at one bank (e.g. bank 0), the other bank (e.g. bank 1), which might contain the previously resulting transform, can be read, and the new samples can be stored at this bank. Still, accessing the active bank, i.e. the bank that is currently used by the wavelet processor, will generate an error as a polite indication that this bank is busy. With this technique, while the processor performs the transform on one bank (either bank 0 or bank 1), the next data can be placed on the other bank. Thus, it improves the overall performance by minimizing the delay caused by the data preparation.

Recall the discussion of two different types of PEs, i.e. the resource-aware PEs and the high-performance PEs. In the resource-aware PEs, we only need to load and store two samples every two clock cycles, or better said one sample per clock cycle. Therefore the memory interface is simple. Contrary, in the high-performance PEs, we need to load and store two samples every one clock cycle. Unlike the traditional method that exploits least significant bit to address both samples, the memory access in our wavelet processor is quite different. See **Fig**. 4.30, when a wavelet decomposition takes place, the even and odd samples are in fact neighbour. In a wavelet reconstruction, we can examine the **Fig**. 4.30 from right to left. Low-pass and high-pass components are located in two different banks. Therefore we need two independent ports to support these types of PEs.

Tab. 4.4: Arbitration of the memory bus ownership.

Signal	Processor (Priority=1)	External (Priority=2)
RE[0] WE[0] RADDR[0] WADDR[0] WDATA[0]	active_bank=0	BANK_SEL=0
RE[1] WE[1] RADDR[1] WADDR[1] WDATA[1]	active_bank=1	BANK_SEL=1

4.7.4 Arbiter

As the name implies, arbiter controls the ownership of the bus. In our architecture, it regulates the ownership of the memory bus. Because there are only two interfaces that access the bus, i.e. the external interface and the processor (which is split into source and sink blocks), the arbitration of the bus is straightforward. The processor has higher priority and the external interface is grant to have the bus access if only if the bank that is accessed is not used by the processor. Error will be reported if the external interface tries to access the active bank. **Tab**. 4.4 summarizes the arbitration priority of the memory. The common memory interfaces are given here and the *active_bank* is controlled by the *FSM*.

4.7.5 Source and Sink

These blocks generate and automatically increment the read and write addresses. The *source* reads data from the memory and transfers it to the PEs. The *sink* reads data from the PEs and writes it to the memory. A special case is considered when performing transforms that are longer than the available PEs. During the in-between transform, in case of forward transform, the *sink* will write the data (which corresponds to the intermediate results) to the memory in adjacent manner (resulting L-H-L-H-...). During the final transform, the *sink* writes the LP and the HP signals into two different pages (resulting L-L-...-H-H-...). The similar handling is also performed by the *source* when performing the inverse transform. To access the correct page, two address masks are used. The first mask is responsible for the data indexing, and the second mask is responsible for the page indexing.

The FSM of the source block is relatively simple. **Fig**. 4.31 depicts the state diagram of the source. When the source receives the *run* signal from the *main FSM*, it determines

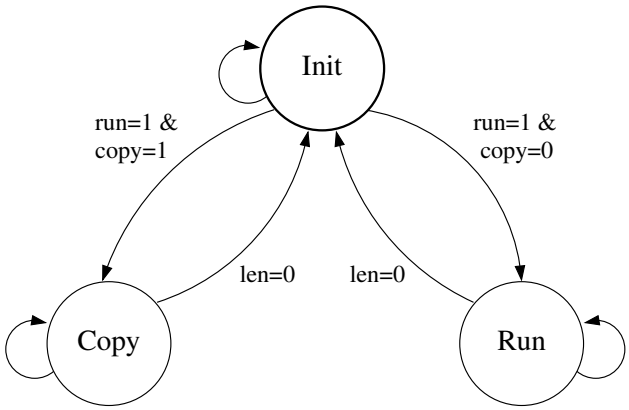


Fig. 4.31: Source FSM.

its operation. If the copy mode is clear, the source starts its normal operation, i.e. it prepares the read address, sets the read enable signal, reads samples the memory and forward them to the top of the PE. The internal counter keeps track about the amount of the samples need to be read. Similar case is applied in case the copy mode is set. The only difference is that the samples are not forwarded to the PE.

As previously stated, source block has two addressing operations for dealing with in-between transform ad the final transform. The masking addressing technique, which is not only simple, but also fast, is used to deliver these features. This can be easily explained using example. Let us assume that the starting address given by the *main FSM* is $addr_c = b1110$ (i.e. two samples from the periodicity extension) and we are now computing the second transform iteration, which makes $mask_1 = b0111$. The address of the first sample is given by $addr = addr_c \wedge mask_1 = b0110$, which is exactly the 6-th sample from the current bank. Additionally, this source address is incremented every clock cycle. By applying the same technique, we will never escape from this bank. When the wavelet reconstruction is performed, the low-pass and high-pass components are located at two different but neighbouring banks. Thus, with similar technique and by using different mask $mask_2 = b1000$, the bank that contains the high-pass component can be accessed by $addr = addr_c \vee mask_2$. Lastly, recall the discussion of the different types of PEs. There exist two PE categories, i.e. resource-aware and high-performance PEs. In the resource-aware PEs, the access to the memory uses one single data line in order to load the even and odd samples alternately every clock cycle. This approach is very efficient in term of the memory interface. Although two samples have to be fed to the PEs at the same time, because of the time-sharing property exploited by these types of architecture, the PEs can only process samples every two clock cycles. Therefore, one even sample can be loaded first, put it in one register, and on the next clock cycle one odd sample will be loaded. Both samples will be fed to the PEs together every two clock cycles. In case of high-performance PEs, these PEs can process samples as high as two samples per clock cycle. In order to take the advantages offered by these types of PEs, two data lanes are needed to read both even and odd samples every clock cycle. Thus, the chained PEs will always

be occupied optimally. Depending on the realization, the source block will increment its address properly.

The sink block performs similar operations to read the samples from the PE and store them to the memory, and to provide the direct path while copying one memory bank. The sink block receives its delayed *run* signal from the *main FSM* in order to cope with the computation latency introduced by the chained PEs.

4.7.6 Latency Counter

This block delays the *run* signal from the *main FSM* to initiate the sink process. The delay amount is different for every lifting steps and it is defined in the *config* block. In case of a memory copy is requested, the latency counter is set to zero.

4.8 Details of the Transform Process

Along with the discussion of the different types of PE architectures, the rests of the components such as the main FSM that control the processor, the memory that is used as a storage, the source that provides samples to the PEs, the sink that retrieves resulting transforms from the PEs and stores them to the memory, have been detailed. Because the transform processes are quite complex, it will not be easy to describe them just by investigating every single component. In addition to that, our wavelet processor is design to have rich features. It has capabilities to perform 2D or higher dimension transforms as well as 1D transform. Although at the first sight it makes less difference about the transform process of 1D signals and higher dimension signals, it does as a matter of fact require different treatments. Also, our wavelet processor is designed to be flexible in the way that it is capable of performing various wavelet transforms with different wavelet filters, based on their lifting scheme representations. In doing so, we have provided a framework by implementing atom function of a prediction or an update inside the PE in a flexible manner. The sample indices of the predictor or updater are not fixed during the design time and can be freely configured during the run-time, and the coefficient indices are also implemented as a context-based RAM instead of ROM.

Even though the size of the memory is fixed during the design time, the transform can be performed at arbitrary sample length. It is of course not entirely correct, because it is not possible to compute the transforms that exceed the size of the memory itself. Thus the transform is limited to have a maximum length of the size of the memory. Also, because of the octave scaling, the length of the sample must satisfy the 2^n-rule.

Taking into account that we are targeting different classes of applications, we are not limiting the architecture by only designing the wavelet processor that is based on the fixed-point arithmetic. The floating-point arithmetic comes into play in order to deliver greater computation flexibility. Not only the different issues regarding the number rep-

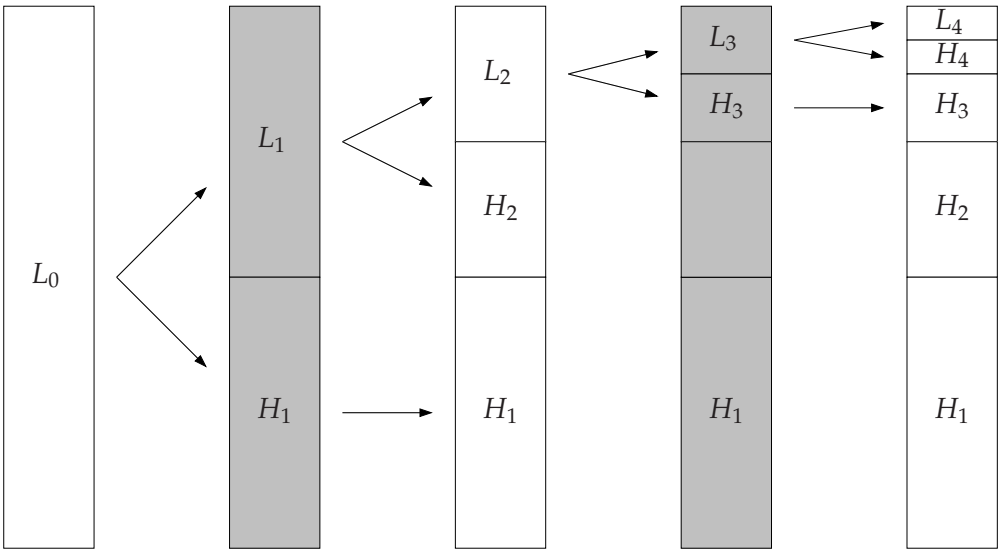


Fig. 4.32: 4-level 1D DWT wavelet decomposition.

resentations are covered, we also design a wavelet processor that can perform discrete wavelet decomposition as well as discrete wavelet reconstruction, and discrete wavelet transform as well as discrete wavelet packet. High-performance versions of the processor are also available to increase the throughput and the computation performance of the wavelet processor with extra costs in area and also complexity of the memory interface.

This section details possible transform processes in order to understand the process of the transform carried out by our wavelet processor. We discuss some possible schemes of wavelet decomposition and wavelet reconstruction for both DWT and DWP and how they inherit similarity.

4.8.1 1D Discrete Wavelet Decomposition for DWT

The first and simple example of the transform process that is supported by our wavelet processor is a 4-level DWT wavelet decomposition. **Fig**. 4.32 illustrates the case. The white boxes represent the primary bank and the grey boxes represent the shadow bank. In this example, the lifting steps are assumed to fit into the available PEs. During the setup, the data is prepared and stored in one bank, in this case the primary bank. This bank is write-only and its shadow is read-only. When the transform is initiated, this state is reversed, which makes the primary bank is read-only and its shadow write-only. The arbiter grants the memory bus to the source and the sink and any access from the external interface to this bank pair will generate an error. For each decomposition, the source reads the written data from the primary bank in an incremental way and forwards them to the top of the chained PEs. Controlled by the latency counter, the sink starts its process a bit later in order to compensate the computation delay introduced by the PEs. The sink writes the resulting transforms to the shadow bank into two different pages. Once the whole samples are processed, the state of the primary and shadow banks is reversed

Tab. 4.5: Initialization values.

Name	Value	Name	Value	Name	Value	Name	Value
$addr_{csrc}$	b0000	off_{src}	b1110	$mask_{src1}$	b1111	$mask_{src2}$	b0000
$addr_{csink}$	b0000	off_{sink}	b0000	$mask_{sink1}$	b0111	$mask_{sink2}$	b1000

again. The same process is again performed for the next decomposition with the length of the processing samples is roughly half of the previous decomposition stage. At the first decomposition stage, because the whole samples are involved, no copy transfer is needed. In contrast, the second and higher decomposition stages require selective copy transfer to be carried on. The goal of the copy transfer is to synchronize the resulting high-pass components on both banks. Additionally, we also improve the performance by removing unnecessary copy transfers. Thus, because the target bank of the final decomposition stage is the primary bank in case of a 4-level wavelet decomposition, during the second decomposition stage, the first resulting high-pass component, denoted as H_1 needs to be copied from the shadow bank to the primary bank. Because H_2 is already stored in the primary bank, which is the final bank, it is not necessary to copy H_2 from the primary bank to the shadow bank during the third decomposition stage. Contrary, H_3 needs to be copied from the shadow bank to the primary bank, and so on.

Up to now we have not discussed details regarding the periodicity extension. Let us assume that 2 past samples are required to compute the transform. When the transform is about to be started, the main FSM initializes the source and the sink addresses and also their offsets and their masks, the values are given in **Tab**. 4.5. Note that without loss of generalities, we use 4-bit address line to illustrate the transform process.

The source fetches its first address by using its offset to localize the sample and its mask to localize the source page, i.e.

$$addr_{src}[0] = (addr_{csrc} \vee off_{src}) \wedge mask_{src1} \tag{4.36}$$

and its next addresses by simple increment, i.e.

$$addr_{src}[n+1] = (addr_{src}[n] + 1) \wedge mask_{src1}, \qquad n \in \mathbb{Z} \tag{4.37}$$

which lead to the following sequential addresses: $b1110$, $b1111$, $b0000$, $b0001$, etc. Note that we use the indexed bracket to emphasize the updating sequence of the address. The first two addresses correspond to the roll-over samples and the rest correspond to the normal samples. The sink fetches its first address in a similar way, i.e.

$$addr_{sink}[0] = (addr_{csink} \vee off_{sink}) \wedge mask_{sink1}, \qquad n \in \mathbb{Z} \tag{4.38}$$

and its next addresses in an alternating incremental paging mode, i.e.

$$addr_{sink}[2n+1] = addr_{sink}[2n] \vee mask_{sink2} \tag{4.39}$$

$$addr_{sink}[2n+2] = (addr_{sink}[2n+1] + 1) \wedge mask_{sink1} \tag{4.40}$$

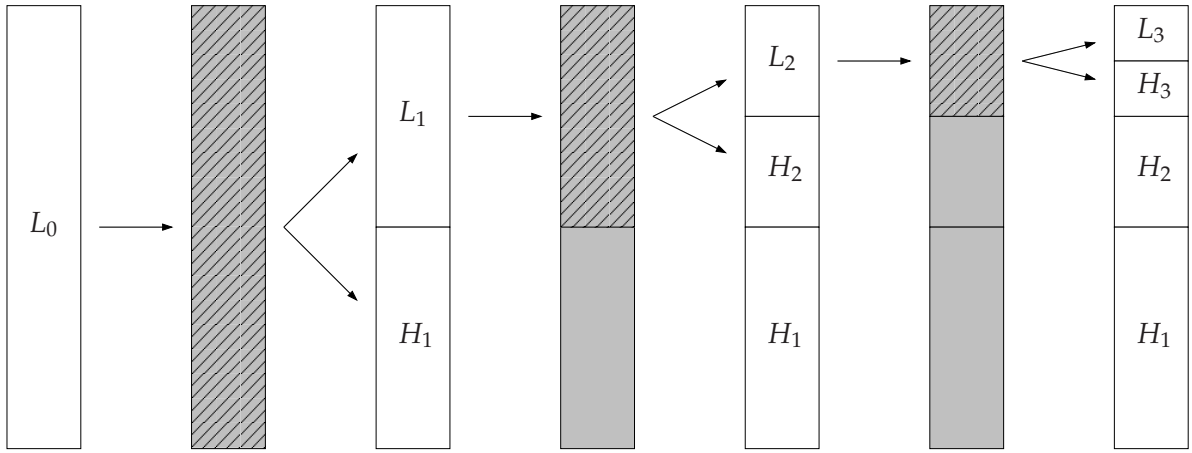


Fig. 4.33: 3-level 1D DWT wavelet decomposition with multiple lifting steps.

which lead to the alternating addresses: $b0000$, $b1000$, $b0001$, $b1001$, etc. Using this two masking technique, the source and the sink can generate their address autonomously. Most of the lifting schemes require previous and future samples. Therefore the number of the processing samples is always greater than the number of the samples itself. We exploit the periodicity extension to solve the problem on computing the transform on the boundary regions (the lower and the upper regions). Nevertheless, we do not implement additional memory or storage to buffer these samples. Instead, we let the processor provides these samples. This solution slightly adds the complexity in the main FSM, but it eliminates the need of the buffers and it is in fact very efficient and flexible. We can address any starting sample (whether it is with a positive or negative offset) without any limitation, which would be the case if an extra buffer is in use.

After the first decomposition stage finishes, the next decomposition stage can be performed by first updating the address masks. As a matter of fact, this can be done in an easy way. We only need to shift the address masks one bit to the right for each decomposition. For example, the setup for the second decomposition will have $mask_{src1} = b0111$, $mask_{src2} = b0000$, $mask_{sink1} = b0011$, $mask_{sink2} = b0100$.

Recall the discussion on taking the advantage of the lifting schemes by storing the temporary results in the memory from **Sec**. 4.2.1 and depicted in **Fig**. 4.3. This method is also supported by our wavelet processor in order to perform wavelet transforms that utilize longer lifting schemes that occupy more than the available PEs. The basic principle of the wavelet decomposition detailed previously is also applied here. **Fig**. 4.33 depicts the scheme of a 3-level 1D DWT wavelet decomposition that uses two lifting groups. Because the term in-place is used to define the result of each atom function, we use the term *in-between* to denote the temporary group lifting results. The diagonal pattern in the figure represents the in-between transform. During the in-between transform, the sink writes its transform outputs in an adjacent manner, resulting L-H-L-H-... form in the shadow bank. When the final transform takes place, the sink perform its regular writing into two different banks to achieve L-L-...-H-H-... form, which is exactly the same as

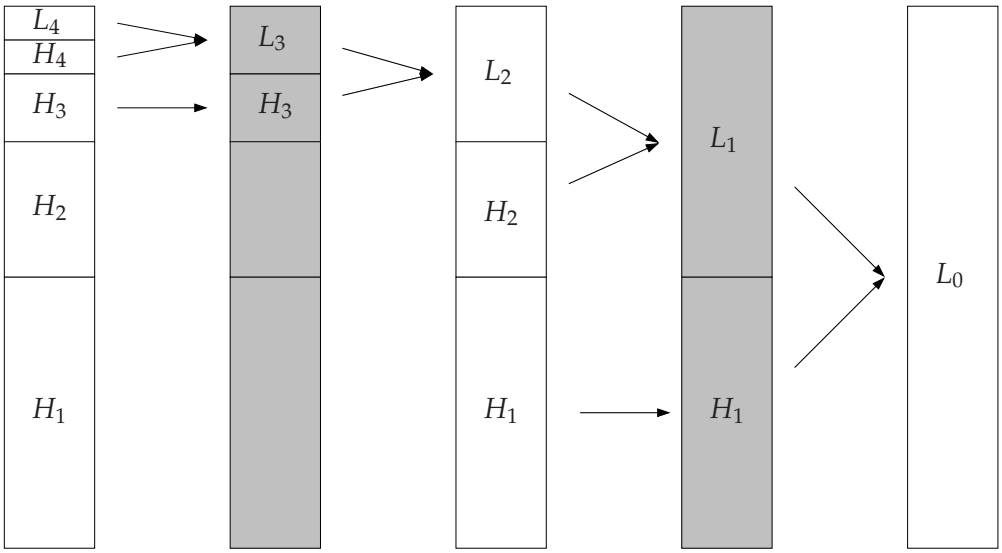


Fig. 4.34: 4-level 1D DWT wavelet reconstruction.

in the previous case. Note that no copy transfer is required here to copy the resulting high-pass components to another bank because the results of each decomposition are stored back into the same primary bank. Nevertheless, if the number of the group lifting belongs to an odd number, this is not the case and the copy transfer is required. The processor automatically checks whether the copy transfer is needed and perform the copy transfer after each decomposition level, the same mechanism as found on the traditional decomposition.

4.8.2 1D Discrete Wavelet Reconstruction for DWT

Because some applications also require to able to reconstruct the original signal back after some further processing, we also implement this feature into our wavelet processor. The scheme on performing the reconstruction is basically the same as on performing the decomposition. To illustrate the reconstruction process, we detail a 4-level DWT reconstruction for a 1D signal. **Fig**. 4.34 depicts the transform process. In this example, it is assumed that the lifting schemes can be fit into available PEs, as it is also the case in the first example of the wavelet decomposition detailed a while ago and the samples are prepared in the primary bank. During the first wavelet reconstruction phase, the low-pass component L_4 and the high-pass component H_4 are located on the different pages in the same primary bank. The source resolves the addresses of these two pages by applying the similar masking technique as described earlier for resolving the address at the sink. It reads the data from both pages and feds them to the PEs. After some delay, the sink starts its process. It reads the resulting transforms from the PEs, and writes them to the shadow bank in a adjacent manner, resulting the higher resolution representation of the signal, L_3. Because L_3 is stored in the shadow bank, we need to copy the high-pass component H_3 from the primary bank to the shadow bank for the next reconstruction stage.

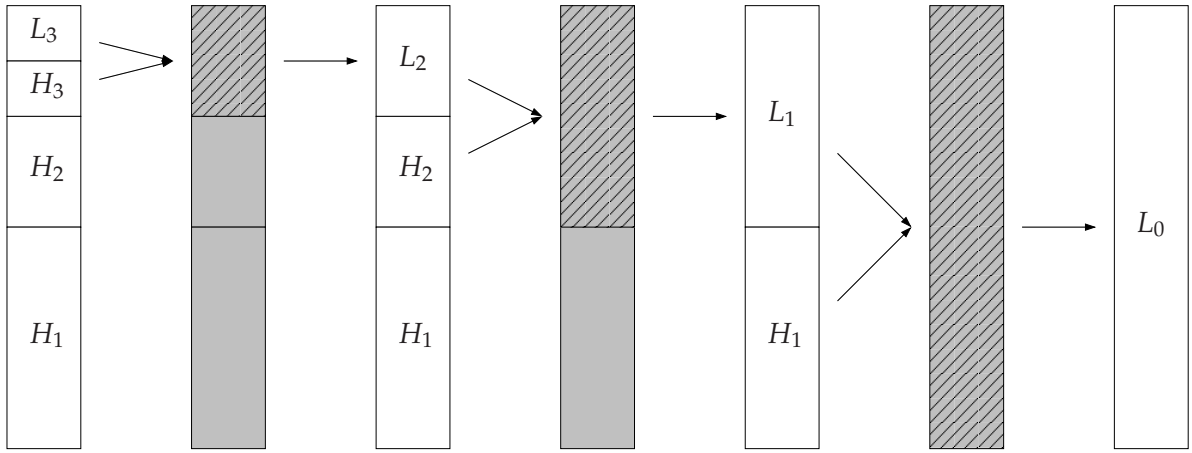


Fig. 4.35: 3-level 1D DWT wavelet reconstruction with multiple lifting steps.

After the copy process finishes, the state of the primary and shadow banks are reversed. This corresponds to the read-only state for the shadow bank and write-only state for the primary bank. Before starting the second reconstruction step, the masks for both source and sink are adjusted. Here, one-bit shift right is applied to cover a larger page for the next reconstruction. Because the results of the second stage reconstruction are stored in same bank where the H_2 is located, it is not necessary to synchronize the high-pass component H_2. The same processes are performed until the whole samples are processed and reconstructed into L_0.

In a 1D decomposition, the initialization values for masks and offset lengths are defined straightforward because the decomposition always starts at the first level and after each decomposition level, we only need to perform one-bit right shift operation to adjust these values for the next decomposition. In contrary, a 1D reconstruction has different starting masks and offset lengths depending on the number of reconstruction level. To address this issue, we implement small look-up tables which contain all possible offsets and masks of all covered level. We can clarify that this is in fact very small look-up tables. Let us assume that the processor has a capability to store 512 samples, i.e. 2^9 samples. The depth of the look-up tables in this case is 9+1 at maximum. It is also possible to limit the depth of the look-up tables with a consideration that wavelet transform is rarely used to decompose the whole signal into its lowest resolution.

In case of a lifting scheme that exercises lifting steps that are larger than the numbers of the available PEs in the processor, we use the same technique by first computing the in-between transforms. **Fig**. 4.35 depicts a 3-level DWT wavelet reconstruction for a 1D signal as an example to illustrate the reconstruction process that uses this feature. During the first in-between transform of each reconstruction phase, the source reads the low-pass and high-pass components pagewise and the sink writes the in-between transform results in adjacent manner. Because of this mechanism, the in-between results can be treated as an alternating even and odd samples. Therefore, during the rest of the in-between transforms, similar to the writing mechanism on the sink, the source reads the samples

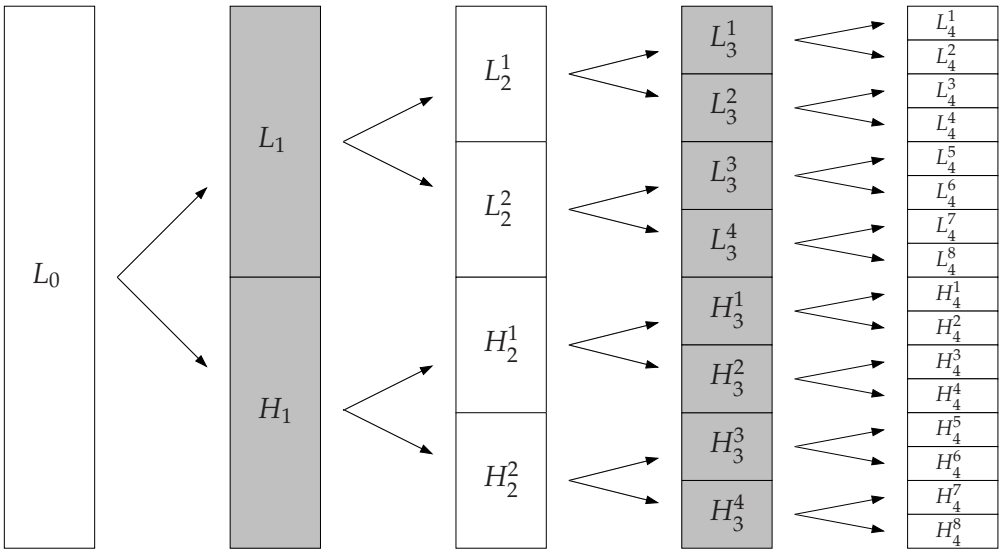


Fig. 4.36: 4-level 1D DWP wavelet decomposition.

in a normal adjacent way. As we can notice that for the lifting schemes that involve even number of lifting groups, it is not necessary to copy any high-pass components due to the fact that the resulting transforms are always stored in the primary bank. This is not the case for the lifting schemes with group odd number where the selective copy transfers will be performed during the final transform of each reconstruction stage, the same as we can find in the previous example for decomposition with multiple lifting steps.

4.8.3 1D Discrete Wavelet Decomposition for DWP

Whereas DWT iterates only the low-pass components to achieve the octave spaced frequency bands representation in the multiresolution signal analysis space, DWP exploits also the resulting high-pass components in order to create an equally spaced frequency bands on the multiresolution signal analysis space. To give a clear understanding how the DWP decomposition is performed by our wavelet processor, we use the same example detailed in the DWT decomposition. **Fig**. 4.36 depicts the transform process of a 4-level DWP decomposition of a 1D signal. The basic principles used in the DWT are applied here. The source reads the samples in an incremental way and the sink writes the resulting transforms into two different pages. Whereas 1-level DWP decomposition is exactly the same as 1-level DWT decomposition, starting from 3-level decomposition, DWP expands the tree equally. Additionally, because DWP always processes the whole samples, no copy transfer is needed to synchronize one or more resulting components. The term L_i and H_i are still used to be consistent with the previous examples, while for higher decomposition level we use different naming to index the resulting low-pass and high-pass components. We do not use the terms LL_i, LH_i, HL_i, LH_i, etc. because these terms are normally used for multidimensional transforms. As we can notice from the figure, the number of processed bands increases two folds for each decomposition stage and

the number of processed samples decreases roughly two folds for each frequency band. The total number of processes samples increases slightly after each decomposition stage due to the fact that the past and future samples with periodicity extension are involved during the computation. We introduce additional mask to address the correct page of the wavelet packets. Thus the source address generations from **Eq**. (4.36) and **Eq**. (4.37) become

$$addr_{src}[0] = (addr_{csrc} \vee off_{src}) \wedge mask_{src1} \vee mask_{dwp} \tag{4.41}$$

$$addr_{src}[n+1] = (addr_{src}[n] + 1) \wedge mask_{src1} \vee mask_{dwp} \tag{4.42}$$

and the sink address generations from **Eq**. (4.38)–**Eq**. (4.40) become

$$addr_{sink}[0] = (addr_{csink} \vee off_{sink}) \wedge mask_{sink1} \vee mask_{dwp} \tag{4.43}$$

$$addr_{sink}[2n+1] = addr_{sink}[2n] \vee mask_{sink2} \vee mask_{dwp} \tag{4.44}$$

$$addr_{sink}[2n+2] = (addr_{sink}[2n+1] + 1) \wedge mask_{sink1} \vee mask_{dwp} \tag{4.45}$$

Updating this mask is also straightforward. Using the same example as before with 4-bit address bus, the initial mask for the DWP is given by $mask_{dwp} = b0000$. Note that the original length of the processed samples, i.e. the length without the offset for periodicity extension, for the first decomposition stage is given by $len = b10000$ which corresponds to the whole samples. The original length of the next decomposition stages is easily computed by again right shifting by one bit operation. Thus, for the second decomposition stage, it is set to $len = b01000$. After finishing the first decomposition on one frequency band on the second decomposition stage (in this case the low-pass component L_1), the mask for DWP is updated by

$$mask_{dwp} = mask_{dwp} + len \tag{4.46}$$

This is in fact an effective solution, which gives the following masks for the second decomposition stage: $b0000$, $b1000$, and for the third stage: $b0000$, $b0100$, $b1000$, $b1100$, and so on, which compute the target pages correctly.

Additionally, we also improve the performance on computing the DWP decomposition by reducing the latency introduced by the data preparation. As we stated earlier, we use periodicity extension in order to address the issue regarding computing the transform on the boundary regions. This means that rollover upper samples are needed to compute the transform on the lower boundary region and rollover lower samples are needed to compute the transform on the upper boundary region. Because the PE is based on a pipeline architecture, we expect computation delay between the first input and the first valid output of the chained PEs. **Fig**. 4.37 shows the timing diagram of the DWP decomposition from second and third stages. The grey boxes represent the shadow bank and the white boxes represent the primary bank. They are drawn exactly to follow the DWP decomposition from **Fig**. 4.36. The cross diagonal patterns represent the rollover upper and lower samples of the same bank. At t_1 the wavelet processor starts to feed the PEs by

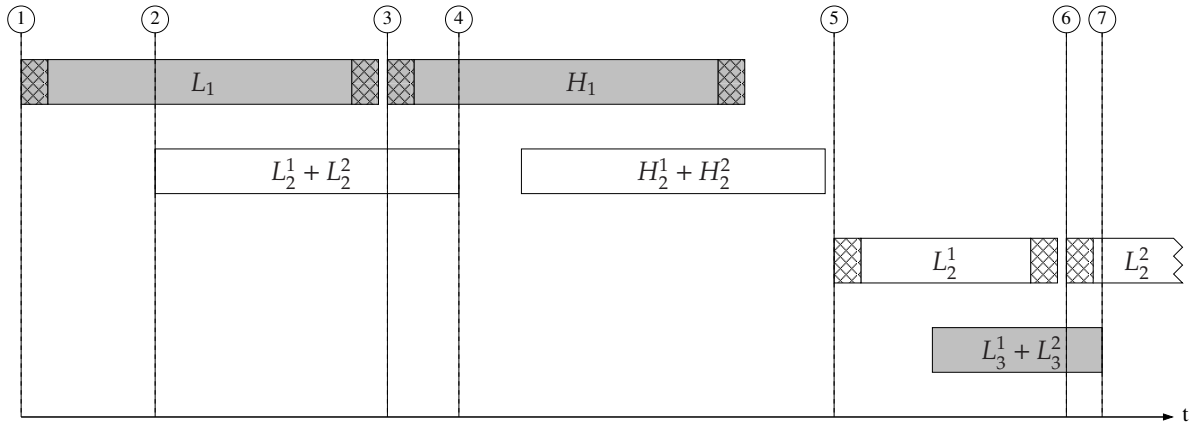


Fig. 4.37: Timing diagram of 1D DWP decomposition.

first preparing the upper rollover samples, followed by the normal samples, and ended by the lower rollover samples. The distance between $t_2 - t_1$ is the computational latency introduced by the PEs. At t_4 the whole resulting transforms of low-pass component L_1, i.e. L_2^1 and L_2^2, are computed and stored in the primary bank. In normal circumstances, the transform of the next band, i.e. the high-pass component H_1, can be started at t_4. Because lifting steps are performed in-place by the PEs, we take this advantage further by preparing the samples from H_1 at t_3 directly after the last sample from L_1. Thus it reduces the latency as much as $t_4 - t_3$ in this case. After the high-pass component H_1 is completely processed, the next decomposition stage can be started at t_5. It is not possible to overlap the data preparation here. The reason is that the next decomposition requires switching the state of the banks which will make the primary bank as a read-only bank and the shadow bank as write-only. This can only be achieved or better said, performed, if all the last resulting transform of the current decomposition stage are stored. The same method is also applied while finishing the transform of component L_2^1. Instead of starting the next band transform of component L_2^2 at t_7, we can feed those samples as early as t_6, which reduces the latency with the amount of $t_7 - t_6$. Note that this optimization is only valid for DWP decomposition and reconstruction. In case of DWT transform, the state of the banks is always switching every time the one decomposition or one reconstruction stage finishes. That is why in the block diagram of the wavelet processor shown in **Fig**. 4.27, the main FSM has two *done* inputs, one from the sink and one from the source. These finish indications are used by the main FSM to perform the preloading of the next frequency band if the condition is satisfied.

We follow the same method from DWT decompositions with multiple liftings in order to compute the DWP decompositions that also require number of PEs that cannot be fit into the available PEs. **Fig**. 4.38 depicts the scheme of a 3-level DWP wavelet decomposition of a 1D signal used as an illustration. The first decomposition stage of a DWP wavelet decomposition that exercises multiple lifting groups is exactly the same as in DWT. The differences are found starting from the second decomposition stage. Previously, we discussed the optimization on DWP decomposition by overlapping the samples. Instead

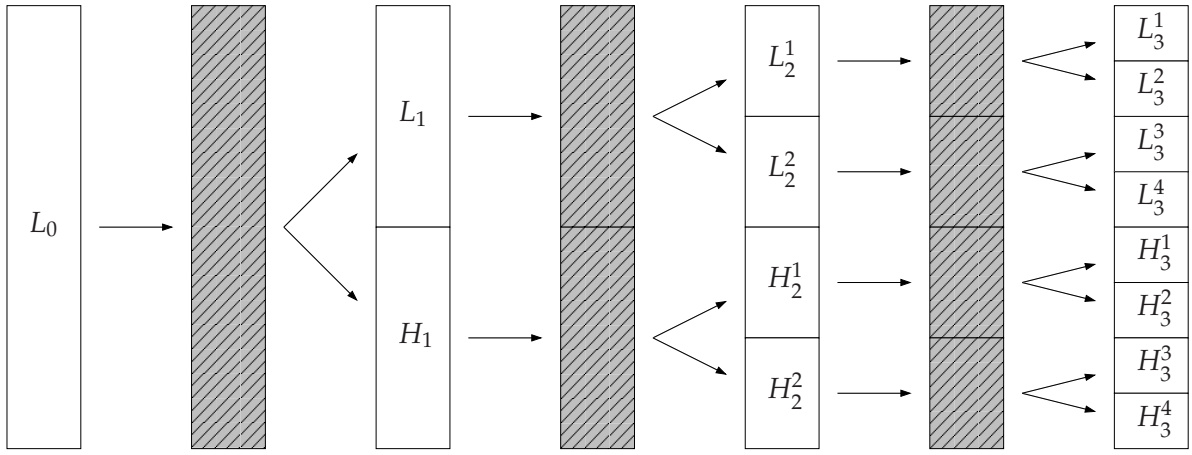


Fig. 4.38: 3-level 1D DWP wavelet decomposition with multiple lifting steps.

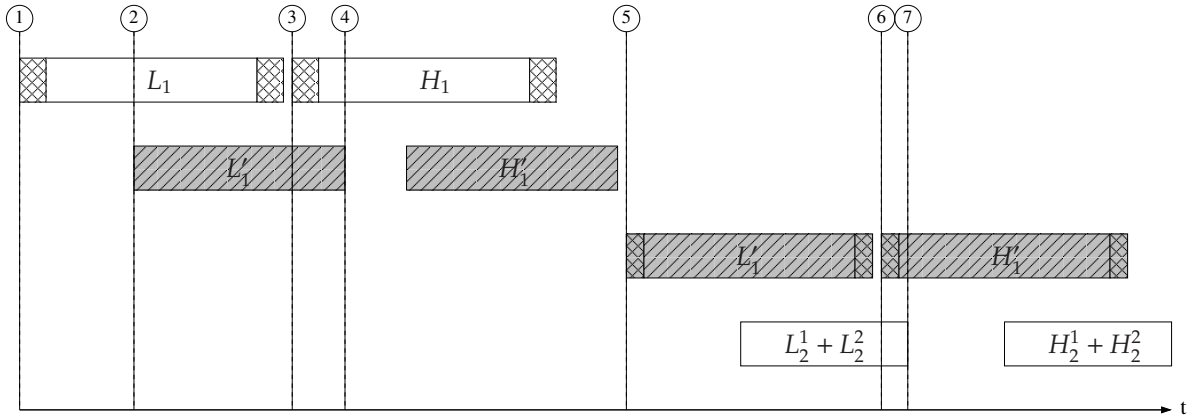


Fig. 4.39: Timing diagram of 1D DWP decomposition with multiple lifting steps.

of completing the transform of each frequency band and continuing with the next frequency band, e.g. decompose L_1 into L_2^1 and L_2^2 followed by H_1 into H_2^1 and H_2^2, we need to complete the in-between transform processes of all involved frequency bands before we start performing the next in-between transforms in order to take the same advantage, e.g. computing the in-between transforms of L_1 and H_1, resulting the in-between transform L_1' and H_1', followed by their second in-between transforms, etc. The reason behind this restriction is the same as discussed previously. Overlapping can only be executed when the state of the banks is not changing. **Fig**. 4.39 shows the timing diagram of this case.

4.8.4 1D Discrete Wavelet Reconstruction for DWP

Fig. 4.40 illustrates the DWP reconstruction process of a 1D signal. As an illustration, we use exactly the same example as before. The same principle as the DWT reconstruction process is also applied here in order to reconstruct the signal. Nevertheless, whole frequency bands are involved from the beginning to reconstruct larger frequency bands in the DWP wavelet reconstruction. Therefore, no copy process is performed because of this

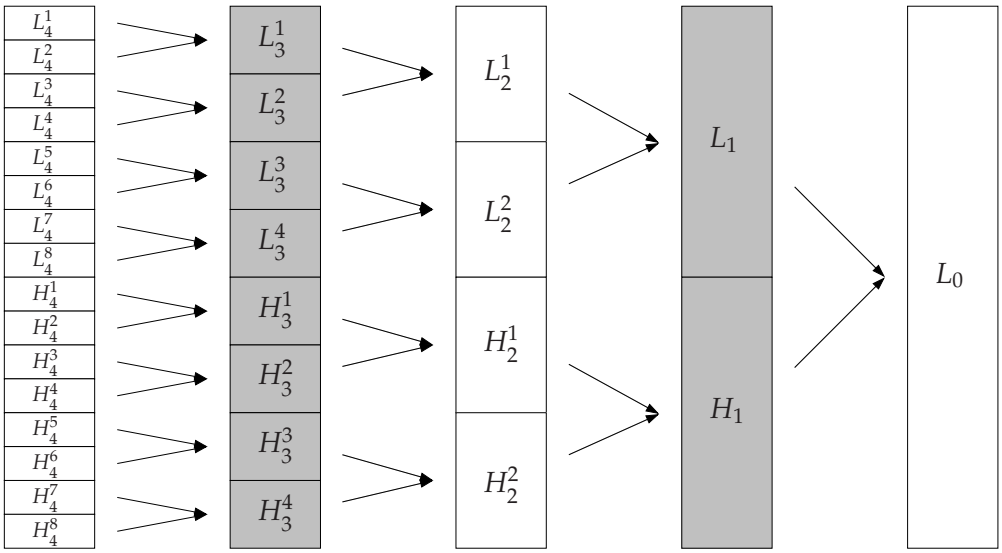


Fig. 4.40: 4-level 1D DWP wavelet reconstruction.

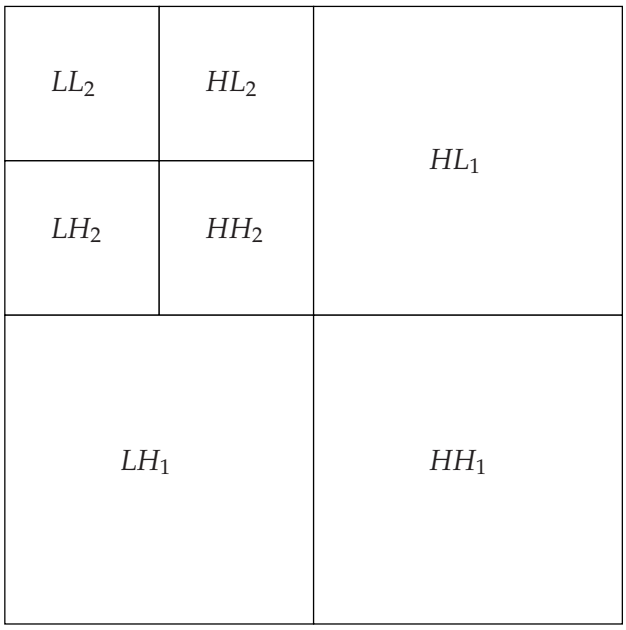


Fig. 4.41: 2-level 2D DWT decomposition.

reason. Additional mask for DWP is also used here to determine and also to localize the two-bank pair that is being transformed. The DWP reconstructions using multiple lifting steps are also supported by our architecture. The transform process is similar to the previous reconstruction examples and it is achieved by computing the in-between transforms first, followed by the final transform on each reconstruction stage.

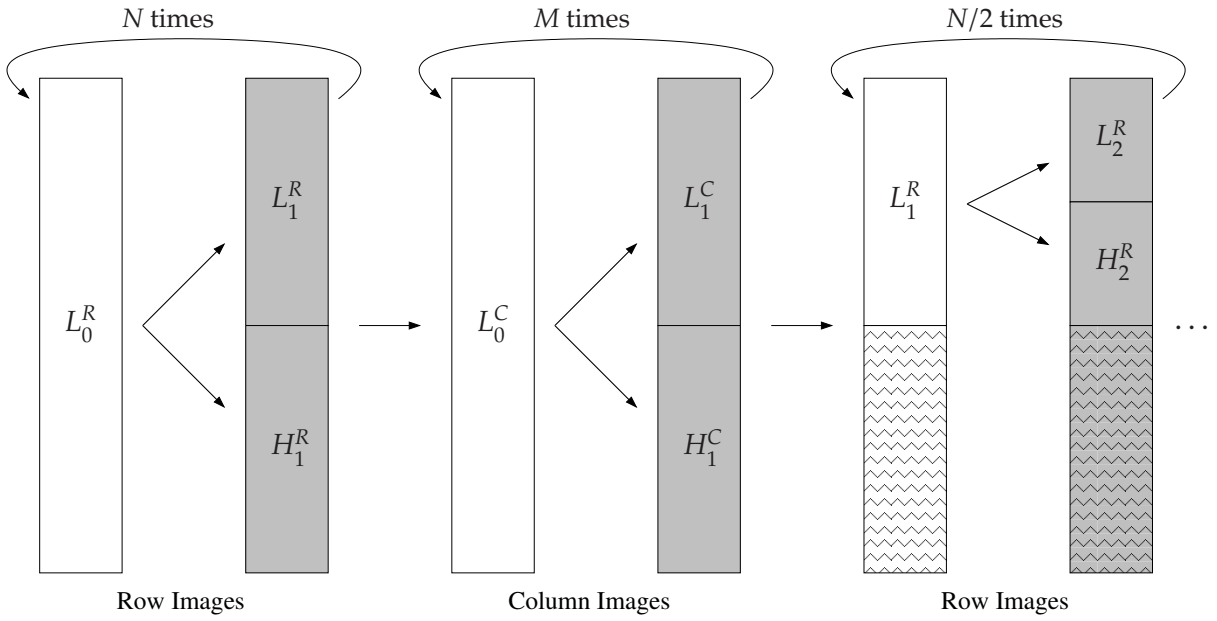


Fig. 4.42: Process of a N-level 2D DWT decomposition.

4.8.5 N-Dimensional DWT Decomposition and Reconstruction

Multidimensional DWT inherits similarities with 1D DWT. Whereas in 1D transform we perform multilevel decomposition by iterating the transform process on the low-pass component and multilevel reconstruction by the same principle applied on the low-pass and high-pass component pairs, in multidimensional transform we want to exploit data dependencies of the signal in all dimensions. Therefore, performing multilevel decomposition on one dimension of the whole signal, followed by another multilevel decomposition on the other dimension will not be an optimal solution, because the correlations between neighbouring samples are only examined on one dimension and not properly exploited on the other dimension.

To exploit the data dependencies between dimensions, it is necessary to perform one-level transform on one dimension for the whole signal, followed by one-level transform on the other dimension, also for the whole signal, and so on. This step is performed iteratively by taking the resulting low-pass component of this signal as the source for the next iteration. As an example, in case of a 2D signal, the first transform decomposes the signal into four components, namely LL_1, LH_1, HL_1 and HH_1. The second decomposition takes the low-pass component, LL_1, to generate again four components LL_2, LH_2, HL_2, and HH_2. This scheme is depicted in **Fig**. 4.41. Note that the first decomposition components have the size of quarter of the original signal and the second decomposition components have the size of quarter of the first decomposition component and so on. Thus in a 2D signal, the amount of the processed samples reduces four folds at each decomposition.

Noting this behaviour, most of the workloads of the multidimensional DWT transform take place on the sample preparation and storing. The wavelet processor itself performs

only a simple 1D transform in this case. As we stated earlier, our proposed wavelet processor can compute the transform at arbitrary sample length. This feature is in fact used heavily in order to compute the transform of a multidimensional signal. Back to the 2D example, assuming it is a $N \times M$ 2D image, the first decomposition is performed by first transforming the row images, followed by column images. The second decomposition uses half the length of the row images and half the length of the column images, which leads to quarter the size of the processed signal. **Fig**. 4.42 depicts the decomposition transform process of a $N \times M$ 2D image performed by the processor. Row images L_0^R are processed first, followed by the column images L_0^C, resulting the low-pass component LL_1. During the second decomposition we only need to prepare half of the samples from the row images and half of the samples from the column images, which corresponds to LL_1. The boxes with triangular wave patterns indicate that these parts of memory are ignored. As we can notice from the figure, we are not limiting ourselves to have an equal length for all dimensions in our architecture. Therefore, it is indeed very flexible because we do not constraint the applications to have equal length. Furthermore, in order to reduce the sample preparation and storing mentioned at the beginning, which is the most workloads in the multidimensional transforms, we expand our storage facility in our wavelet processor by providing two dedicated banks, called bank 0 and bank 1, with each bank consists of one primary and one shadow banks, as detailed in **Sec**. 4.7.3. Therefore, while the processor is performing the transform of one row image in one bank (e.g. bank 0), the next row image samples can be prepared on the other bank (e.g. bank 1).

The reconstruction of a multilevel N-dimensional signal also follows the same technique as in the reconstruction for 1D signal where the reconstruction is performed one-level by one-level and one dimension by one dimension, in order to reconstruct the original signal back.

4.8.6 N-Dimensional DWP Decomposition and Reconstruction

The N-dimensional decomposition and reconstruction using DWP is more complex compared to the N-dimensional DWT decomposition and reconstruction. It is due to the fact that although the number of samples decreases four folds for each DWP decomposition as it is also the case in a N-dimensional DWT transform, the number of processed banks increases four folds. Because of the periodicity extension, the total number of processed samples also increases every decomposition stage, depending on the wavelet filters used for the transform. Additionally, the complexity grows because the transform is also executed one-level by one-level and one dimension by one dimension in order to exploit the correlations in all directions, the same as in the N-dimensional DWT transforms. Let us assume that we already computed the first level DWP decomposition of a 2D image as depicted in the left part of **Fig**. 4.43 and we want compute the second level DWP decomposition as depicted in the right part of the figure. Note that first level DWP decomposition is exactly the same as first level DWT decomposition for both 1D and multidimensional

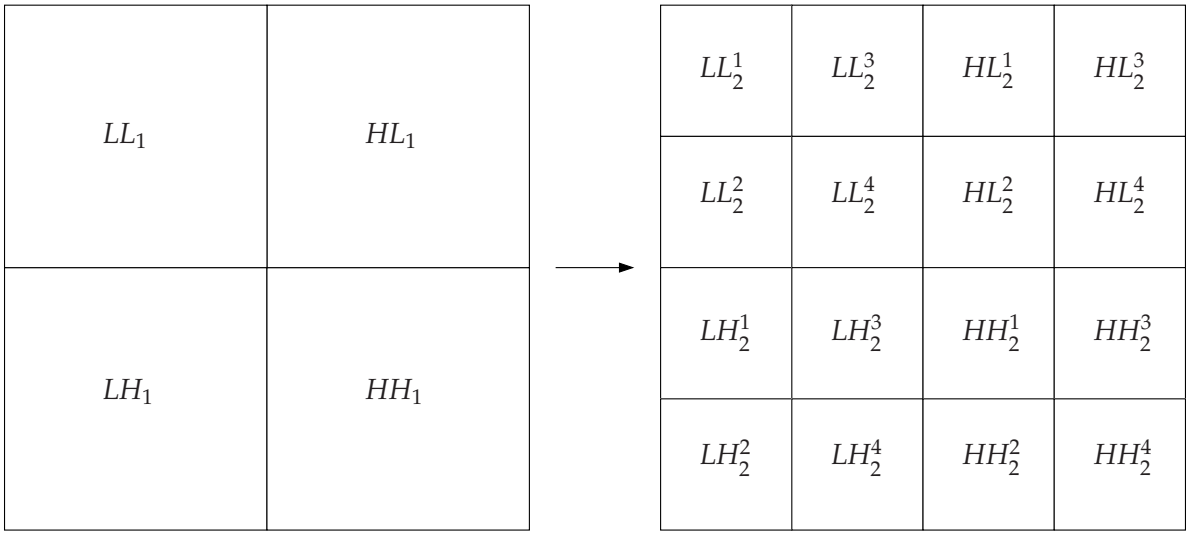


Fig. 4.43: First level and second level 2D DWP decomposition.

transforms. To decompose the image one level higher, that is the second decomposition stage of the signal, we need to provide not only the low-pass component LL_1, but also the high-pass components LH_1, HL_1, and HH_1. In other words, all resulting components from the previous decomposition are involved in order to decompose the signal one level higher. This is as a matter of fact will be the advantage of our architecture. Instead of decomposing each component independently one after another, which is also possible in our wavelet processor by treating every single component as one independent component and applying DWT decomposition for each component, we can improve the performance by treating these components as one. We only need to inform the processor that the involved samples consist of two frequency components in this case, thus the processor can use the overlapping feature detailed in **Sec**. 4.8.3 and depicted in **Fig**. 4.37. Additionally, similar to the multilevel N-dimensional decomposition, because we do not constraint the signal to have the sample sizes in every dimension, the architecture is indeed flexible.

The multilevel N-dimensional reconstruction using DWP is also similar to the DWT case. Here, we also treat all decomposed frequency components as one and let the processor takes the advantage of the scheduling by using the overlapping technique.

One final remark regarding the multilevel 1D and N-dimensional DWP decompositions and also reconstruction is that it is also possible to have different wavelet filters to decompose each frequency band. The only restriction is that each band needs to be decomposed independently. Thus, the overlapping method cannot be applied here. The same principle is also applied in reconstructing the signal. We need to use the corresponding synthesis filter pair to reconstruct each bank independently, in order to achieve perfect reconstruction.

4.9 Concluding Remarks

In this chapter, the design of our wavelet processors was detailed. The discussion covered the architectures of four different PEs that are used as a prediction or an update atom function to compute the lifting step from the lifting scheme. We discussed also the context switch that is used to compute wavelet transforms that exercise longer wavelet filters, memory interface and its bank organization, main FSM that controls the processor, etc. Additionally, the processes of several possible transforms were detailed to describe how these transforms are carried out by our processor and to show capabilities of our processor on performing DWT and DWP transforms.

Chapter 5

Analysis on Performance and Benchmarks

Contents

5.1 Analysis on Floating-Point Arithmetic Cores 127

5.1.1 2-Stage Floating-Point Multiplier 127

5.1.2 4-Stage Floating-Point Adder 128

5.1.3 3-Stage Floating-Point Adder 130

5.1.4 Discussion on the Floating-Point Cores 131

5.2 Analysis on Different Types of PEs 132

5.3 Analysis on Different Number of PEs 136

5.4 Analysis on Different Memory Organizations 138

5.5 Analysis on Supporting Wavelet Packet 140

5.6 Performances 142

5.6.1 Performances on 1D Signals 142

5.6.2 Performances on 2D Signals 148

5.7 Benchmarks 151

5.8 Comparisons 155

5.9 Concluding Remarks 157

As stated in the previous chapter, we maintain the design of our wavelet processor to be flexible. These flexibilities do not only involve the capabilities of the processor to perform wavelet transforms with various wavelet filters based on their lifting scheme representation, or the feature the processor provides in supporting DWT decomposition and reconstruction, as well as DWP decomposition and reconstruction, and 1D signal as well as N-dimensional signal. The flexibility of our wavelet processor extends to the

types of the PEs that can be used in our wavelet processor, i.e. resource-aware PEs, high-performance PEs, fixed-point PEs, and floating-point PEs, in order to suit the application's demands. The architecture of our wavelet processor does not restrict the size of the memory that can be implemented in our processor. The memory size in the wavelet transform plays a role in determining the maximum length of the samples that can be transformed by the processor without needing to break them into several segments or tiles. Therefore, by extending our design to accept arbitrary size of memory, we can target more applications.

The flexibilities also cover the data width of the samples that can be selected and realized. The fixed-point representation of the number can also be chosen during the design time to determine how many bits will be used to represent the integer part and how many bits will be used to represent the fractional part. Additionally, our floating-point arithmetic cores also deliver greater flexibilities. They not only support 32-bit single-precision and 64-bit double-precision, but they can be customized with different sizes of mantissa and different sizes of exponent.

This chapter discusses in detail about the performances of our wavelet processor. Our processor is written in VHDL and is based on the modular and parametric approach to make the design adaptable and to deliver such flexibility. There are various configurations that can be selected and optimized during the design time to adapt the targeting applications. Because fixed-point arithmetic and floating-point arithmetic cannot be joined together without needing additional wrapper, which will consume additional logics, we decide to have either wavelet processor that is based on fixed-point arithmetic or wavelet processor that is based on floating-point arithmetic directly during the design time. Also, we implement a centralized controller that organizes the chained PEs. Thus, it allows us to simplify the architecture of the PEs. Because the PEs are designed to be simple, in the way that each PE does not need FSM attached on it to control the behaviour of the PE, the resource-aware PEs that exploit the time-sharing feature cannot be put together with the high-performance PEs that utilize two multipliers and two adders. Although it is feasible, this kind of mixing does not deliver any advantage, while the chained PEs work based on the slowest ones, i.e. the resource-aware versions.

The number of PEs that can be integrated into the processor is freely chosen during the design time. Therefore we can trade-off between the required chip area and the performance. Smaller number of PEs means that more in-between transforms will take place in order to compute wavelet filters that have longer lifting scheme representation. This also means that the processor requires more time to compute the transforms. In contrary, larger number of PEs contribute to larger chip area, and also some degradation of the achievable operating frequency due to the interconnections. The size of the contexts can also be chosen during the design to adapt the applications need. In order to reduce the size of the look-up tables that are used for maskings, we can also set the maximum allowable decomposition or reconstruction level that the wavelet processor can handle.

Independent from the size of the samples, we also provide two options for the mem-

ory organization, i.e. 1×2 banks and 2×2 banks, that can be integrated into the processor with the later is used to decrease the data preparation time. As we will detail shortly, we also investigate that integrating DWP decomposition and reconstruction features in the wavelet processor consumes additional chip area and slightly decreases the achievable operating frequency. Taking into account that most applications require only DWT transforms, we can enable the feature of DWP decomposition and reconstruction in the design time.

Because the processor is designed to offer greater flexibilities, we provide analysis on several components that are integrated in the processor. **Sec**. 5.1 details the analysis of the floating-point arithmetics used in the floating-point-based PEs. **Sec**. 5.2 discusses the impact of different types of PEs, starting from resource-aware and high-performance fixed-point PEs, and followed by their floating-point counterparts. **Sec**. 5.3 covers the analysis of increasing the number of PEs employed in the processor and **Sec**. 5.4 details the analysis of exploiting additional banks in order to increase the throughput. Because not all applications require the DWP decomposition and reconstruction features, it is by default disabled. **Sec**. 5.5 examines the impact of adding DWP in our wavelet processor. **Sec**. 5.6 explains the performance of our wavelet processors for 1D and 2D signals and **Sec**. 5.7 details their corresponding benchmarks. **Sec**. 5.8 gives a comparison with other existing architectures and **Sec**. 5.9 concludes this chapter.

5.1 Analysis on Floating-Point Arithmetic Cores

The hardware realization of the fixed-point arithmetics used in fixed-point PEs is straightforward despite the diversities in data width implementation. Additionally, we use the generic integer-based multiplier and adder to realize both fixed-point arithmetic functions with additional shifting to realign the result. In other words, we let the underlying CAD tools to optimize these arithmetic functions. In contrary, although the same principles are used to realize the floating-point multiplier and adder, some design considerations need to be investigated to make them customizable, in order to adapt with different data widths and also different floating-point representations. Besides the single-precision and double-precision formats which are well defined, our floating-point arithmetics can be set to have different representations, not only in term of the data widths, but also in term of the number of bits used for mantissa, the number of bits used for exponents, and the number of bits used as guard bits. All of these configurations are supported by our floating-point arithmetic architectures.

5.1.1 2-Stage Floating-Point Multiplier

In the previous chapter, we stated that there exist abundant floating-point multiplier architectures. For the sake of the simplicity of the design, we utilize the relatively straight-

Tab. 5.1: Estimated area and frequency of the floating-point multiplier.

Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Frequency (in MHz)	Remarks
16-bit	31250	92.20 %	641.03	10-bit mantissa, 5-bit exponent
20-bit	40704	92.93 %	512.82	14-bit mantissa, 5-bit exponent
24-bit	48381	93.08 %	398.41	18-bit mantissa, 5-bit exponent
28-bit	92397	95.85 %	362.32	21-bit mantissa, 6-bit exponent
32-bit	98103	95.52 %	353.36	23-bit mantissa, 8-bit exponent
48-bit	166309	96.23 %	208.33	38-bit mantissa, 9-bit exponent
64-bit	244675	96.65 %	132.63	52-bit mantissa, 11-bit exponent

forward implementation of the floating-point multiplier by using two-stage architecture to distribute the critical paths. Because we are targeting the middle-range floating-point resolution, i.e. up to 32-bit single-precision format, one can directly oversee the performance degradation if the architecture is synthesized for higher data width. To justify the performance of our floating-point multiplier, we synthesize the multiplier with different configurations. Our floating-point multiplier supports two types of rounding, i.e. rounding to zero and rounding to nearest. Since rounding to nearest delivers a better rounding mechanism with slightly increase in the chip area, we use rounding to nearest with three guard bits on all floating-point arithmetics. The model is simulated with ModelSim provided by ModelTech in order to verify the functionalities of the floating-point multiplier. We use 0.18-μm technology with the UMC library in our process. Design Vision from Synopsys is used to synthesize the architecture. **Tab**. 5.1 lists the synthesis results of the estimated area and the estimated operating speed of our floating-point multiplier with different data widths. **Fig**. 5.1 shows the relation of the achievable operating speed and the estimated area with various data widths. Five different data widths ranging from 16-bit up to 32-bit are detailed intensively because those are within our range of interest. Additionally, two floating-point representations, i.e. 48-bit and 64-bit double-precision, are also given. As we mentioned earlier, our floating-point multiplier architecture is not efficient for the high-resolution representations. It is caused by the integer multiplier that uses only one stage to compute its result. Nonetheless, because we are focusing on the lower resolution representation, i.e. up to 32-bit single-precision, the performance of the floating-point multiplier with two stages is sufficient. For the non-standard formats, the parameters for mantissa and exponent are chosen to provide higher precision.

5.1.2 4-Stage Floating-Point Adder

Besides the floating-point multiplier, we also design two dedicated floating-point adders that can accept three inputs. In this section, we detail the performance of the 4-stage floating-point adder with three inputs. The same configuration as in the floating-point

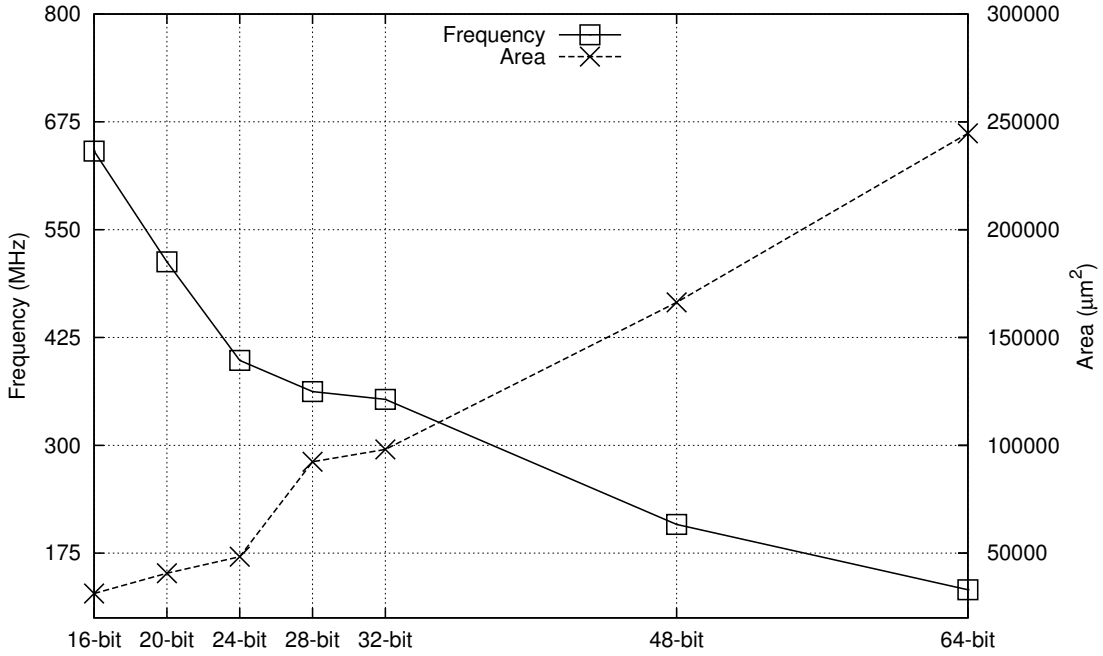


Fig. 5.1: Estimated frequency and area of floating-point multiplier for various data widths.

Tab. 5.2: Estimated area and frequency of the 4-stage floating-point adder.

Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Frequency (in MHz)
16-bit	38262	82.73 %	714.29
20-bit	47752	83.19 %	657.89
24-bit	52213	81.88 %	552.49
28-bit	71911	84.72 %	520.83
32-bit	77114	84.25 %	507.61
48-bit	115370	84.74 %	458.72
64-bit	160215	85.60 %	434.78

multiplier is taken. The model is also verified with ModelSim and the design is synthesized with UMC 0.18-μm process. To ease the configuration and also the integration with our wavelet processor, we also provide two different rounding mechanisms in our floating-point adder, i.e. rounding to zero and rounding to nearest. Similarly, rounding to nearest, which provides better rounding performance which slightly adds up the number of logics, is used. Three guard bits are also taken to guard the arithmetic operations inside the adder. **Tab**. 5.2 summarizes the synthesis results of different data width implementations of our 4-stage 3-input floating-point adder and **Fig**. 5.2 depicts the trend of the frequency versus area used by the adder. Note that we also use the same floating-point format for each data width as in the floating-point multiplier.

As we can see from the table and also from the figure, the achievable operating speed

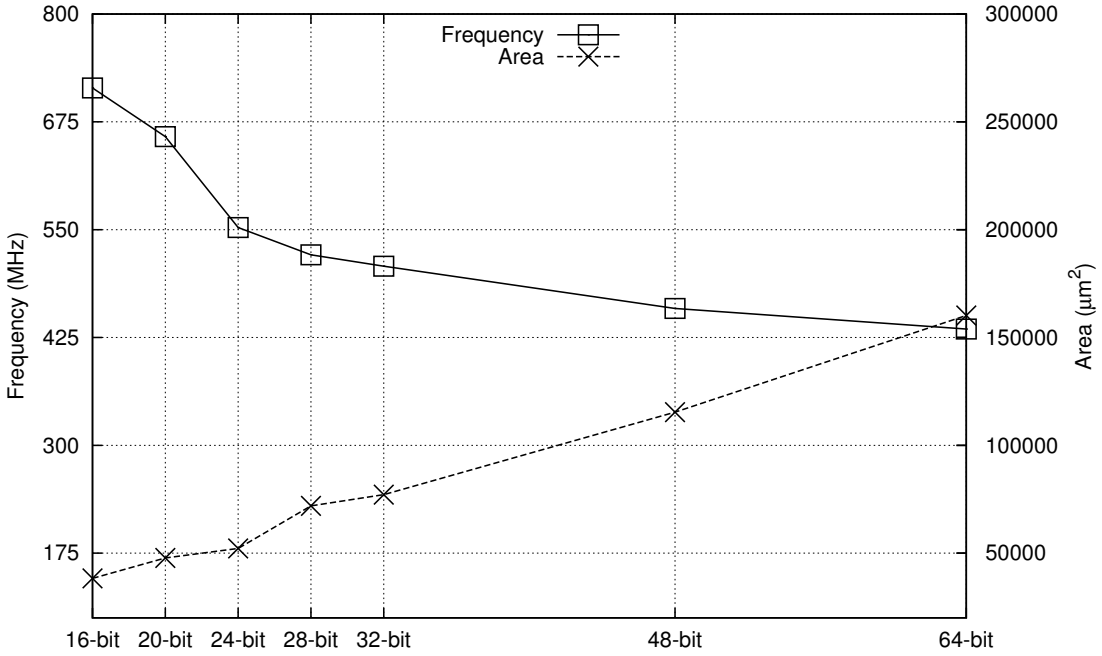


Fig. 5.2: Estimated frequency and area of 4-stage floating-point adder for various data widths.

Tab. 5.3: Estimated area and frequency of the 3-stage floating-point adder.

Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Frequency (in MHz)
16-bit	49236	87.85 %	574.71
20-bit	52329	86.51 %	490.20
24-bit	74927	89.16 %	431.03
28-bit	90984	90.01 %	423.73
32-bit	99003	89.70 %	414.94
48-bit	174153	91.74 %	377.36
64-bit	231575	92.02 %	359.71

of the adder is quite linear in the higher data width range. In the lower region, the performance is mostly affected by the exponent processing blocks that are chained with shift block and add/sub block. The area consumption is linearly increased, similar as in the normal adder.

5.1.3 3-Stage Floating-Point Adder

To couple with the high-performance architecture where three inputs are available at the same time and they have to be processed at the same clock cycle, we also optimize the design of our 3-input floating-point adder. Contrary to the 4-stage floating-point adder where the third input is fed at the next clock cycle, the number of pipelines in this archi-

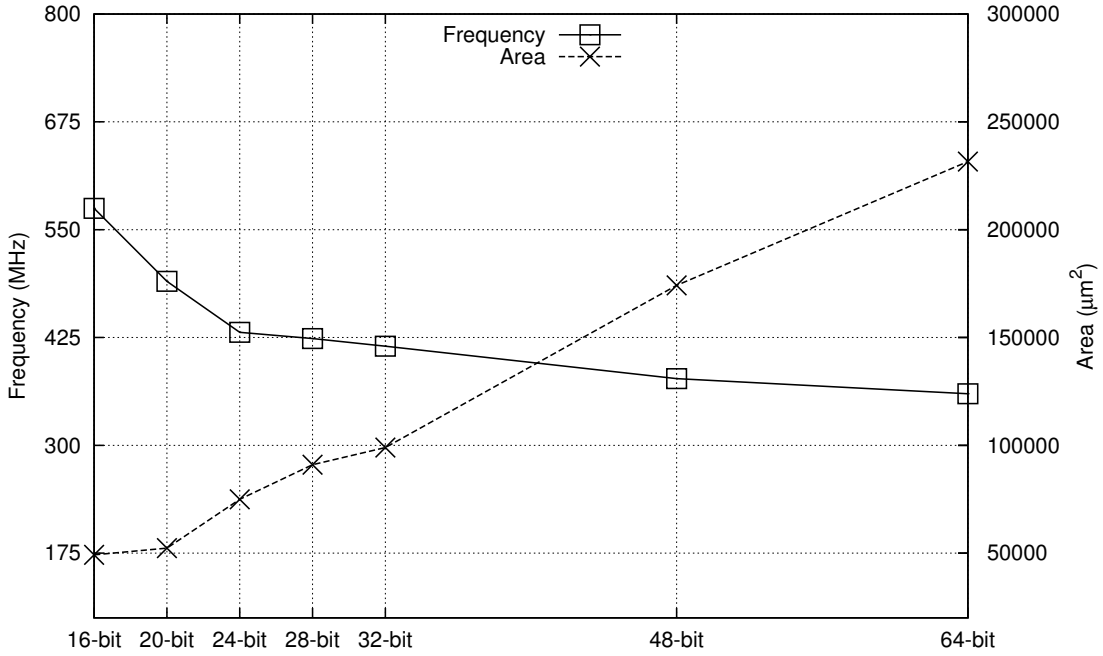


Fig. 5.3: Estimated frequency and area of 3-stage floating-point adder for various data widths.

tecture is reduced into three and all three inputs are processed at the same clock cycle. As one can directly notice, reducing the number of pipelines contributes to the lower achievable operating speed, which is also the case here. **Tab**. 5.3 summarizes the estimated area and frequency of our 3-stage 3-input floating-point adder and **Fig**. 5.3 shows the performance and area versus the data width. The design is also synthesized with the same 0.18-μm process.

5.1.4 Discussion on the Floating-Point Cores

Multiplier and adder are the arithmetic operations used in our PE to compute the prediction and the update of a lifting step. Therefore, the performance of the PE depends mainly on the performance of these arithmetic operations. **Fig**. 5.4 plots the achievable operating speed for each data width for our 2-stage floating-point multiplier and 4-stage and 3-stage floating-point adders to give us a better overview of the performance limit of the PEs that are based on floating-point arithmetics.

Because our range of interest is between 16-bit and 32-bit, we restrict our discussions within this range. Contrary to the realization of the resource-aware floating-point PEs which is limited by the performance of the floating-point multiplier, the achievable operating speed of high-performance floating-point PEs is affected by both multiplier and adder. For the low data widths, the achievable performance is mainly affected by the floating-point adder whereas for the middle range, the floating-point multiplier will be the major bottleneck in achieving a higher operating speed. Additionally, we must not ignore the fact that the realization of high-performance PEs requires two floating-point

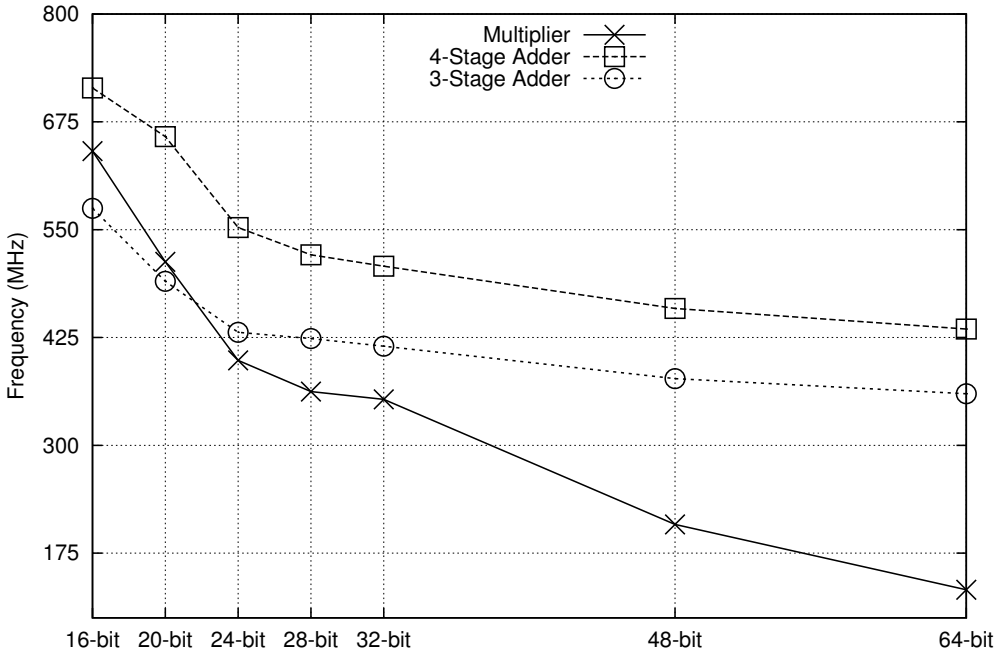


Fig. 5.4: Estimated frequency of floating-point arithmetics.

Tab. 5.4: Fixed-point mapping.

Data Width	Shifter
16-bit	9
20-bit	13
24-bit	17
28-bit	21
32-bit	25

multipliers, which will increase the chip area.

5.2 Analysis on Different Types of PEs

The analysis of the arithmetic cores alone does not tell us much about the performance of the PE. For this purpose, we synthesize four different types of PEs, i.e. resource-aware fixed-point PEs, high-performance fixed-point PEs, resource-aware floating-point PEs, and high-performance floating-point PEs. Recall also our discussion regarding the normalizer units that are placed at the top and at the bottom of the chained PEs. Because adding normalization to the PEs costs additional multiplexers, and on some PEs it also requires additional FIFO to synchronize the outputs, the chip area used for these PEs is larger than the normal PEs and the achievable operating speed decreases slightly. We also provide synthesis results of the PEs that include this feature to give us knowledge about

Tab. 5.5: Estimated area and frequency of the normal PEs and the PEs with normalizer that are based on fixed-point arithmetics.

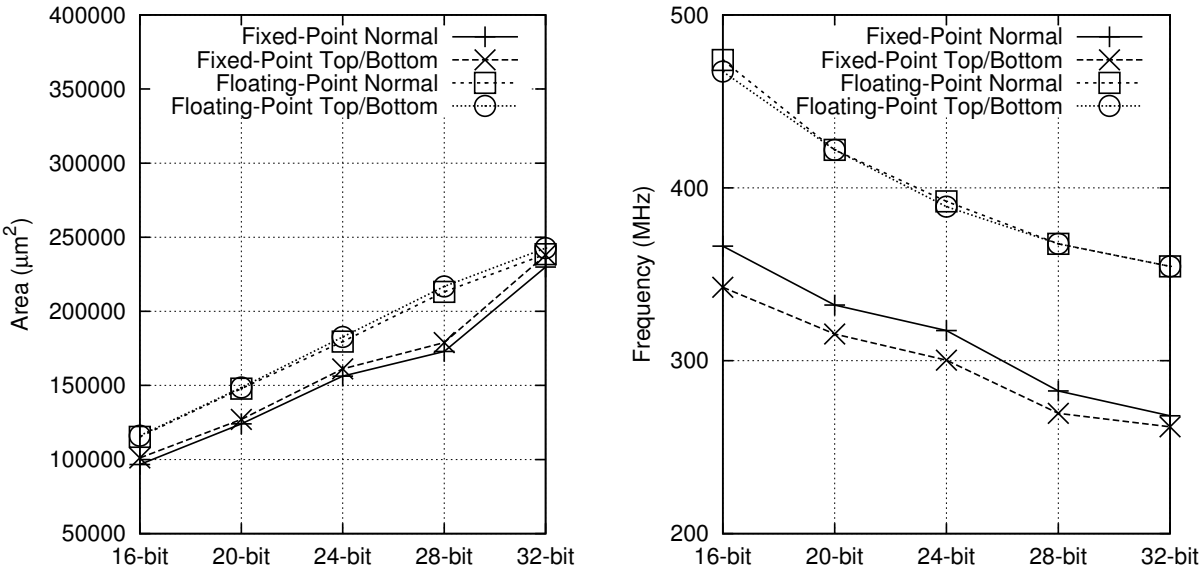
Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		Normal PE			Top/Bottom PE		
Resource Aware	16-bit	96626	41.81 %	366.30	101039	43.78 %	342.47
	20-bit	124050	45.24 %	332.23	127111	46.05 %	315.46
	24-bit	156310	49.06 %	317.46	161135	50.18 %	300.30
	28-bit	172966	47.31 %	282.49	178918	48.67 %	269.54
	32-bit	229847	55.18 %	268.10	238040	56.39 %	261.78
		Normal PE			Top/Bottom PE		
High Performance	16-bit	134508	49.20 %	362.32	138914	50.29 %	354.61
	20-bit	177725	53.25 %	335.57	178424	52.99 %	330.03
	24-bit	224854	56.47 %	317.46	228186	56.79 %	310.56
	28-bit	252991	55.52 %	287.35	257645	56.03 %	277.78
	32-bit	359671	64.57 %	271.74	370502	65.42 %	266.67

the increase of chip area and decrease of operating speed of these types of PEs. **Tab**. 5.5 summarizes the synthesis report of the fixed-point PEs, including the PEs with the normalizer unit, i.e. the PEs that are placed at the top and at the bottom of the chained PEs. Likewise, **Tab**. 5.6 summarizes the synthesis report of the floating-point PEs, including also the PEs with the normalizer unit. The designs are synthesized using UMC 0.18-μm process. For fixed-point architectures, the fixed-point locations are set during the design time and given in **Tab**. 5.4. Note that increasing the data width in fixed-point architectures does not always mean that the sample range that can be processed by the PEs also increases. Here, we focus on processing 8-bit samples. The value of shifter increases with increasing the data width in order to deliver higher precision. It tells us the amount of the shift to right operation after the integer multiplication. Therefore, in fixed-point implementations, the input samples need to be magnified by performing the shift to left operation with some value during the sample preparation to achieve better signal to noise ratio.

The PE alone cannot be synthesized directly because it needs the context switch that stores the configurations of the PE. The synthesis results shown in **Tab**. 5.5 and **Tab**. 5.6 include also the context switch module. Note that the size of contexts can be freely chosen. For this purpose, we provide 8 available contexts for the PE. Additionally, the maximum depth of unit delays used to store samples from both inputs is also adjustable and for this purpose, 8 unit delays are chosen. **Tab**. 5.7 and **Tab**. 5.8 detail the area consumption of the main components of the PEs. The adder in the fixed-point PEs consumes a very small chip area whereas multiplier requires a much larger area. In contrary, adder and multiplier in the floating-point PEs moderately require same chip area. In the high-performance fixed-

Tab. 5.6: Estimated area and frequency of the normal PEs and the PEs with the normalizer that are based on floating-point arithmetics.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		Normal PE			Top/Bottom PE		
Resource Aware	16-bit	115251	42.17 %	473.93	116122	42.07 %	467.29
	20-bit	147816	45.41 %	421.94	148603	45.19 %	421.94
	24-bit	179556	47.22 %	392.16	182727	47.74 %	389.11
	28-bit	213151	49.05 %	367.65	216838	49.61 %	367.65
	32-bit	238598	48.76 %	354.61	242607	49.30 %	354.61
		Normal PE			Top/Bottom PE		
High Performance	16-bit	151184	49.48 %	401.60	152119	49.33 %	389.10
	20-bit	197078	53.00 %	362.32	199249	53.24 %	352.11
	24-bit	243630	55.52 %	334.45	253348	56.90 %	328.95
	28-bit	309648	59.74 %	318.47	303751	58.75 %	311.53
	32-bit	340462	58.61 %	309.60	352600	59.83 %	306.75

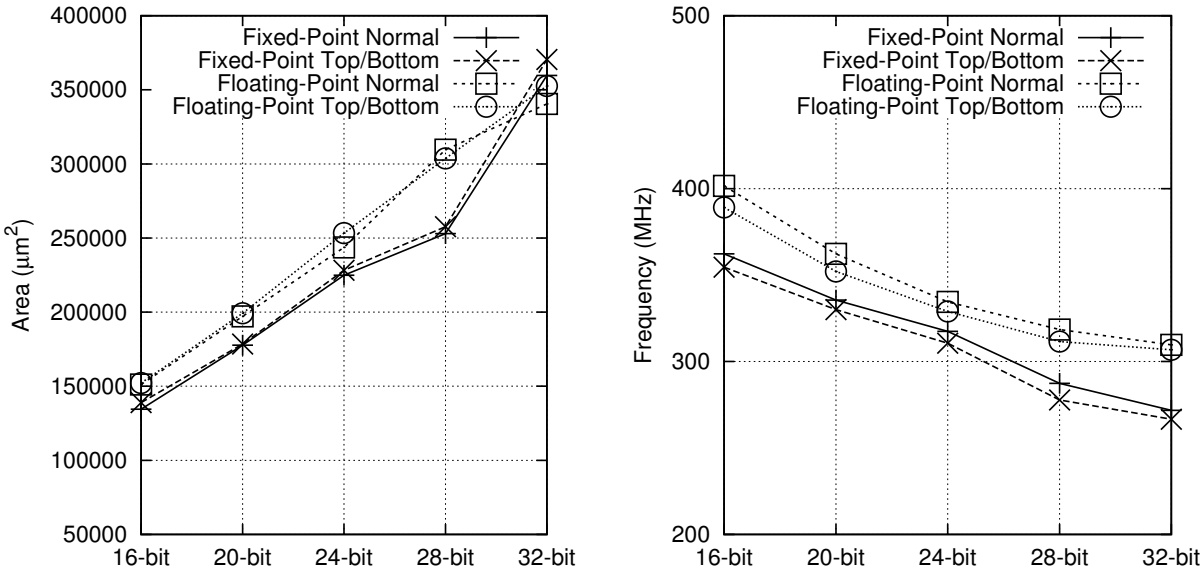


(a) Estimated Area

(b) Estimated Frequency

Fig. 5.5: Estimated area and estimated frequency of resource-aware fixed-point and floating-point PEs.

point and floating-point PEs, two multipliers are used, therefore the area used by the multipliers is two folds compared to the resource-aware PEs. The same reason applies for the fixed-point adder. In contrary, our 3-stage floating-point adder with three inputs for high-performance PEs slightly adds the chip area compared to 4-stage floating-point adder for resource-aware PEs. Other main contributions are context switch and unit de-



(a) Estimated Area

(b) Estimated Frequency

Fig. 5.6: Estimated area and estimated frequency of high-performance fixed-point and floating-point PEs.

Tab. 5.7: Details of area usage of the fixed-point PEs (all values are in μm^2).

Type	D.Width	Mult.	Add.	Context	Delay	Mult.	Add.	Context	Delay
		Normal PE				Top/Bottom PE			
Resource Aware	16-bit	23128	2181	36965	28831	23276	2116	37643	30043
	20-bit	33924	2874	43365	37146	33843	2680	44071	36546
	24-bit	51226	3374	49981	43400	48590	3251	50423	46552
	28-bit	55596	4045	56022	48029	55913	3964	56687	49193
	32-bit	93913	4542	62651	57619	95838	4522	63463	58222
		Normal PE				Top/Bottom PE			
High Performance	16-bit	44471	4312	36833	43610	43217	4267	37540	45868
	20-bit	67096	5383	43252	55439	63573	5342	44117	55264
	24-bit	97742	6571	49838	62870	92084	6525	50471	67150
	28-bit	107122	7854	56512	72334	105810	7883	56780	74054
	32-bit	187822	8970	62683	89807	189241	9057	63512	92149

lays. Depending on the type of the PE and the data width, these components can occupy more than half of the chip area. Therefore, reducing the size of the context switch and the depth of the unit delay will greatly reduce the total chip area, especially when more PEs are employed.

From **Tab**. 5.5 and **Tab**. 5.6, we can notice directly that adding the normalization feature to the PE adds up the chip area. Therefore, these PEs are only implemented at the

Tab. 5.8: Details of area usage of the floating-point PEs (all values are in μm^2).

Type	D.Width	Mult.	Add.	Context	Delay	Mult.	Add.	Context	Delay
		Normal PE				Top/Bottom PE			
Resource Aware	16-bit	16725	28792	36736	24576	15889	28536	37566	24575
	20-bit	27069	36020	43346	30869	25889	35804	44068	30892
	24-bit	36856	43626	49668	36727	37588	43642	50442	36691
	28-bit	49000	51009	55738	42701	49297	51445	56767	42701
	32-bit	54125	56770	62254	48800	54696	56857	63347	48839
		Normal PE				Top/Bottom PE			
High Performance	16-bit	32959	32095	36743	43246	31714	30959	37556	43459
	20-bit	50481	39897	43484	55568	49248	39591	44149	55668
	24-bit	70989	48064	49706	65761	73618	48129	50461	67902
	28-bit	99932	66879	56261	75950	97384	56886	56983	77492
	32-bit	113812	67801	62712	83995	114560	68004	63483	89224

top and at the bottom of the chained PEs. High-performance PEs consume more chip area in both fixed-point and floating-point architectures because two multipliers are used. In the fixed-point implementations, the performance of the PEs that have normalizer unit is lower than the ones without normalizer unit. Nevertheless, this is not always the case in the floating-point implementations. Adding normalizer unit consumes only additional logic on some data width implementations on the PE that are based on floating-point arithmetic.

The fixed-point architectures achieve lower operating speed compared to the floating-point architectures on the same data width. It is not a surprise because the floating-point arithmetics are pipeline-based whereas the fixed-point arithmetics require only one clock cycle to perform their task. One interesting point is that chip areas on resource-aware floating-point architectures do not differ too much from the same fixed-point ones. In the 32-bit implementations, they even consume less chip area. In high-performance architectures, the area consumption of floating-point architectures is even more efficient compared to the fixed-point ones. This happens because in high-performance architectures, the multipliers dominate the chip area if the designs are optimized for speed. In floating-point multiplication, the size of the mantissas are smaller than the size of the fixed-point numbers, e.g. for single-precision floating-point representation, we need to multiply two 23-bit mantissas (plus additional hidden bit and guard bits), whereas in 32-bit fixed-point, we need to multiply two 32-bit integers. **Fig**. 5.5 and **Fig**. 5.6 give better representations of both estimated area and frequency for both architectures. These results also show us that our 3-input floating-point adders are area efficient.

Tab. 5.9: Estimated area and frequency of the fixed-point wavelet processors with different numbers of PEs.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		4 PEs			8 PEs		
Resource Aware	16-bit	1190779	32.82 %	320.51	1613353	33.18 %	319.49
	20-bit	1482630	33.61 %	300.30	2027300	34.95 %	300.30
	24-bit	1786319	34.57 %	283.29	2447721	36.29 %	283.29
	28-bit	2075246	34.80 %	261.10	2833377	36.47 %	259.74
	32-bit	2462826	37.57 %	244.50	3403854	40.05 %	244.50
		4 PEs			8 PEs		
High Performance	16-bit	1705183	50.05 %	346.02	2266971	47.99 %	343.64
	20-bit	2044817	48.71 %	318.47	2750951	47.50 %	316.46
	24-bit	2509156	50.36 %	304.88	3413945	49.91 %	304.88
	28-bit	2907810	50.39 %	280.90	3968071	50.22 %	278.55
	32-bit	3433965	52.25 %	261.78	4606610	51.36 %	261.10

5.3 Analysis on Different Number of PEs

Previous section discusses the performance of four different types of PEs along with the PEs that are placed at the top and at the bottom of our wavelet processor. Each type of PE is synthesized to provide detail information regarding the estimated chip area and also estimated operating speed. The PE alone is in fact only an atom function of a prediction or an update. Therefore, only analyzing the performance of the PEs is not enough. This section deals with the complete package of the wavelet processor. As we detailed in **Chap**. 4, the wavelet processor consists of chained PEs, memory banks, FSM, source and sink, etc. We also stated that our wavelet processor offers greater flexibility, not only in terms of customizable data width and two different arithmetic cores, i.e. fixed-point and floating-point, but also in terms of the size of the memory, the memory organization, the size of the contexts, etc. Because it will not be possible to cover the whole parameter variations that can be applied to realize the processor, we select some of the important parameter configurations to give us a better insight of the processor.

One of the parameters that play an important role in designing the wavelet processor is the number of PEs that will be integrated inside the processor. The processor with higher number of PEs is able to compute longer lifting steps without using the in-between transform feature, which will decrease the computation throughput. Even if the in-between feature must be used in order to compute wavelet transforms that exercise longer lifting steps, the number of in-between transforms that will be carried out will be smaller for the processor with higher number of PEs. Nevertheless, increasing the number of PEs implemented in the processor consumes also additional chip area. Therefore,

Tab. 5.10: Estimated area and frequency of the floating-point wavelet processors with different numbers of PEs.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		4 PEs			8 PEs		
Resource Aware	16-bit	1288972	33.57 %	467.29	1839602	35.27 %	454.55
	20-bit	1609921	34.54 %	420.17	2290049	36.37 %	409.84
	24-bit	1926188	35.05 %	381.68	2752967	37.38 %	380.23
	28-bit	2252560	35.73 %	363.64	3221467	38.26 %	361.01
	32-bit	2559636	35.78 %	354.61	3665776	38.54 %	354.61
		4 PEs			8 PEs		
High Performance	16-bit	1803424	50.22 %	386.10	2452631	48.16 %	384.62
	20-bit	2153434	48.75 %	344.83	2979621	47.82 %	343.64
	24-bit	2603814	49.73 %	321.54	3614205	49.20 %	316.46
	28-bit	3074730	50.72 %	307.69	4345572	51.21 %	307.69
	32-bit	3523820	51.13 %	303.95	4978551	51.68 %	302.11

choosing the right number of PEs will depend strongly on the applications. To justify the performances of the wavelet processors that carry different number of PEs, we synthesize our wavelet processor with different numbers of PEs. Four cases are also considered here to cope with four different types of PEs, i.e. resource-aware fixed-point wavelet processor, high-performance fixed-point wavelet processor, resource-aware floating-point wavelet processor, and high-performance floating-point wavelet processor. Some parameters are fixed in all cases to give a better comparison. The wavelet processors are synthesized using 0.18-μm process, carry 1×2 banks with 512 words, have 16 contexts on each PE, do not support DWP decomposition and reconstruction. Two different numbers of PEs are taken here, i.e. 4 and 8 PEs configuration. **Tab**. 5.9 and **Tab**. 5.10 summarize the synthesis results for both fixed-point and floating-point wavelet processors.

Because of the modular design of the PEs, interconnection demand between PEs are only limited with the neighbouring PEs, and source and sink blocks for the PEs located at the top and at the bottom. Therefore, increasing the number of PEs employed into the processor only increases the required chip area at most instances.

5.4 Analysis on Different Memory Organizations

Besides supporting different types of arithmetics, i.e. fixed-point and floating-point arithmetics, and different numbers of PEs, the size of the memory that can be integrated inside the wavelet processor is also flexible. Nevertheless, we are not focusing on the maximum number of samples our wavelet processor is able to execute the transform without divid-

Tab. 5.11: Estimated area and frequency of the fixed-point wavelet processors with different memory configurations.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		1×2×512			2×2×512		
Resource Aware	16-bit	1613353	33.18 %	319.49	2434805	31.78 %	319.49
	20-bit	2027300	34.95 %	300.30	3038418	32.62 %	300.30
	24-bit	2447721	36.29 %	283.29	3661089	33.52 %	283.29
	28-bit	2833377	36.47 %	259.74	4240929	33.50 %	259.74
	32-bit	3403854	40.05 %	244.50	5036176	36.33 %	244.50
		1×2×512			2×2×512		
High Performance	16-bit	2266971	47.99 %	343.64	3453612	48.94 %	340.14
	20-bit	2750951	47.50 %	316.46	4135395	47.41 %	316.46
	24-bit	3413945	49.91 %	304.88	5045498	48.74 %	303.03
	28-bit	3968071	50.22 %	278.55	5897558	49.17 %	278.55
	32-bit	4606610	51.36 %	261.10	7036980	51.55 %	258.40

Tab. 5.12: Estimated area and frequency of the floating-point wavelet processors with different memory configurations.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		1×2×512			2×2×512		
Resource Aware	16-bit	1839602	35.27 %	454.55	2656361	33.23 %	454.55
	20-bit	2290049	36.37 %	409.84	3313078	34.02 %	409.84
	24-bit	2752967	37.38 %	380.23	3968616	34.52 %	380.23
	28-bit	3221467	38.26 %	361.01	4634999	35.08 %	361.01
	32-bit	3665776	38.54 %	354.61	5268309	35.11 %	352.11
		1×2×512			2×2×512		
High Performance	16-bit	2452631	48.16 %	384.62	3656284	49.24 %	381.68
	20-bit	2979621	47.82 %	343.64	4363582	47.63 %	343.64
	24-bit	3614205	49.20 %	316.46	5266696	48.50 %	315.46
	28-bit	4345572	51.21 %	307.69	6268334	49.86 %	307.69
	32-bit	4978551	51.68 %	302.11	7159382	50.07 %	302.11

ing the samples blockwise. On this analysis, we measure the effects of using two different memory bank configurations, i.e. 1×2 and 2×2 banks, in respect to the chip area and the achievable operating speed.

Same as the previous analysis, four different cases are also examined here, i.e. resource-

Tab. 5.13: Details of area usage of the fixed-point wavelet processors with different memory configurations (all values are in μm^2).

Type	Data Width	PEs	Memory	Others	PEs	Memory	Others
		1×2×512			2×2×512		
Resource Aware	16-bit	735053	816428	62255	738288	1632876	64413
	20-bit	950106	1013528	64153	945751	2027056	66413
	24-bit	1171727	1210638	65996	1172171	2421296	68172
	28-bit	1358100	1407741	68314	1355571	2815482	70514
	32-bit	1729449	1604824	70930	1754767	3209628	72864
		1×2×512			2×2×512		
High Performance	16-bit	1000367	1200322	65994	984493	2400599	72906
	20-bit	1297133	1385272	68802	1294935	2769391	76621
	24-bit	1692220	1650643	74774	1670743	3300602	83981
	28-bit	1976882	1918183	79821	1985112	3836303	80078
	32-bit	2344889	2186339	83810	2586749	4372024	79448

Tab. 5.14: Details of area usage of the floating-point wavelet processors with different memory configurations (all values are in μm^2).

Type	Data Width	PEs	Memory	Others	PEs	Memory	Others
		1×2×512			2×2×512		
Resource Aware	16-bit	958782	816428	64703	957217	1632857	66985
	20-bit	1210369	1013528	66636	1215665	2027056	70811
	24-bit	1474829	1210638	68157	1476878	2421296	71012
	28-bit	1744130	1407722	70951	1747620	2815463	73083
	32-bit	1989156	1604804	71990	1984822	3209628	73863
		1×2×512			2×2×512		
High Performance	16-bit	1185291	1200293	66732	1186164	2400579	73907
	20-bit	1525319	1385020	70709	1522118	2769404	78783
	24-bit	1892492	1650372	76660	1892044	3300602	82588
	28-bit	2352382	1918231	82470	2354557	3836274	78864
	32-bit	2715629	2186048	85665	2707171	4372050	81834

aware fixed-point wavelet processor, high-performance fixed-point wavelet processor, resource-aware floating-point wavelet processor, and high-performance floating-point wavelet processor. Some parameters are also fixed for all the cases and they are given exactly as in the previous analysis with different number of PEs. Here, we also fix the number of PEs to 8.

Details of the area consumption are summarized in **Tab**. 5.13 and **Tab**. 5.14. It is obvi-

Tab. 5.15: Estimated area and frequency of the fixed-point wavelet processors without and with DWP support.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		DWT & IDWT only			DWT, IDWT, DWP, IDWP		
Resource Aware	16-bit	1613353	33.18 %	319.49	1620194	33.22 %	319.49
	20-bit	2027300	34.95 %	300.30	2031306	34.88 %	300.30
	24-bit	2447721	36.29 %	283.29	2461216	36.48 %	283.29
	28-bit	2833377	36.47 %	259.74	2832506	36.31 %	257.73
	32-bit	3403854	40.05 %	244.50	3414316	40.11 %	244.50
		DWT & IDWT only			DWT, IDWT, DWP, IDWP		
High Performance	16-bit	2266971	47.99 %	343.64	2268855	47.86 %	343.64
	20-bit	2750951	47.50 %	316.46	2756589	47.47 %	315.46
	24-bit	3413945	49.91 %	304.88	3428551	50.02 %	304.88
	28-bit	3968071	50.22 %	278.55	3978596	50.26 %	278.55
	32-bit	4606610	51.36 %	261.10	4816143	53.39 %	259.07

ous that memories consume the most of the chip area. Using 2×2 memory banks configuration doubles the area used for memories, while keeping the chip area for the PEs almost constant. One interesting fact drawn from these tables is that the controller occupies very small footprint and does not increase vastly with the increasing of the data width.

5.5 Analysis on Supporting Wavelet Packet

In addition to the DWT decomposition and reconstruction, our wavelet processor has also capability to perform DWP decomposition and reconstruction. Because not all applications require these features, we decided to include these features during the design time. Therefore, for the wavelet processors that are not synthesized with DWP feature on, they can only compute DWT. For the wavelet processors that are synthesized with DWP feature on, they can compute both DWT and DWP, depending on the selected mode.

Tab. 5.15 and **Tab**. 5.16 summarize the estimated area and estimated frequency of the different types of wavelet processors with and without DWP feature for different data widths. Some configurations are fixed and the same as previous implementations in order to give a better justification. 8 contexts are available for the processor, 8 delay units are implemented, the processors have 8 PEs, and 1×2 memory banks configuration is used in order to minimize the synthesis time.

Although DWP decomposition and reconstruction are more complex compared to DWT decomposition and reconstruction, the algorithm to compute DWP itself is basically

Tab. 5.16: Estimated area and frequency of the floating-point wavelet processors without and with DWP support.

Type	Data Width	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)	Est. Area (in μm^2)	% Area for Logic	Est. Freq. (in MHz)
		DWT & IDWT only			DWT, IDWT, DWP, IDWP		
Resource Aware	16-bit	1839602	35.27 %	454.55	1845043	35.25 %	454.55
	20-bit	2290049	36.37 %	409.84	2299893	36.47 %	409.84
	24-bit	2752967	37.38 %	380.23	2757322	37.33 %	380.23
	28-bit	3221467	38.26 %	361.01	3230531	38.31 %	361.01
	32-bit	3665776	38.54 %	354.61	3677050	38.62 %	354.61
		DWT & IDWT only			DWT, IDWT, DWP, IDWP		
High Performance	16-bit	2452631	48.16 %	384.62	2456928	48.10 %	381.68
	20-bit	2979621	47.82 %	343.64	2990836	47.89 %	343.64
	24-bit	3614205	49.20 %	316.46	3631768	49.34 %	316.46
	28-bit	4345572	51.21 %	307.69	4333289	50.98 %	303.03
	32-bit	4978551	51.68 %	302.11	4948876	51.32 %	298.51

the same as DWT. The only difference is that all frequency bands need to be processed. Because the samples and the results are stored in two different memory locations, the iteration for DWP can be scheduled in a simple manner. Additionally, the address masking techniques that we use to compute DWT can be easily extended to compute DWP without significant effort. Therefore, the processors that also support DWP decomposition and reconstruction do not differ too much compared to the processors that support DWT decomposition and reconstruction only. The estimated frequency of the DWP-capable processors is at most of instances the same as the DWT-only processors.

5.6 Performances

In order to realize the fixed-point multiplication between the samples and the coefficients, we utilized an integer multiplier and a shifter to reduce the hardware cost for fixed-point wavelet processors. As the compensation, this implementation leads to errors caused by the rounding of the wavelet coefficients and the cropping of the multiplication results. Although the same issues occur on the floating-point wavelet processors, these issues are relaxed because the floating-point representation offers higher dynamic range.

5.6.1 Performances on 1D Signals

To measure the level of correctness of our design, we perform DWTs/DWPs and their corresponding inverse transforms on some predefined signals. Three different 8-bit full-

swing signals, i.e. sine, sawtooth, and random, which are used as references, are forward and inverse transformed using (9,7), Daub-4, Daub-6, Symlet-6, and Coiflet-2 wavelet filters. We do not use (5,3) wavelet filter because this filter is integer-based filter with integer-based coefficients. Therefore, if we exercise this filter, the reconstructed signals are exactly the same as the original signals. That is why it is used for lossless transform of JPEG2000. The random signal has a uniform distribution.

The lifting step coefficients of these wavelet filters are summarized in **Tab**. 5.17. These coefficients are shortened to save space. In case of fixed-point arithmetics, because the coefficients have to be represented as integers, depending on the data width, they will be magnified with some factor, and the result will be rounded and used as lifting coefficients. In case of floating-point arithmetics, we only need to convert the floating-point number to the target format. ModelSim is used to compare and verify the results. The SNR is computed using:

$$SNR_{(dB)} = 20 \times log_{10}\left(\frac{\sum|signal|}{\sum|signal - result|}\right) \tag{5.1}$$

where *signal* corresponds to the input vector and *result* corresponds to the output of the decomposition and followed by the reconstruction.

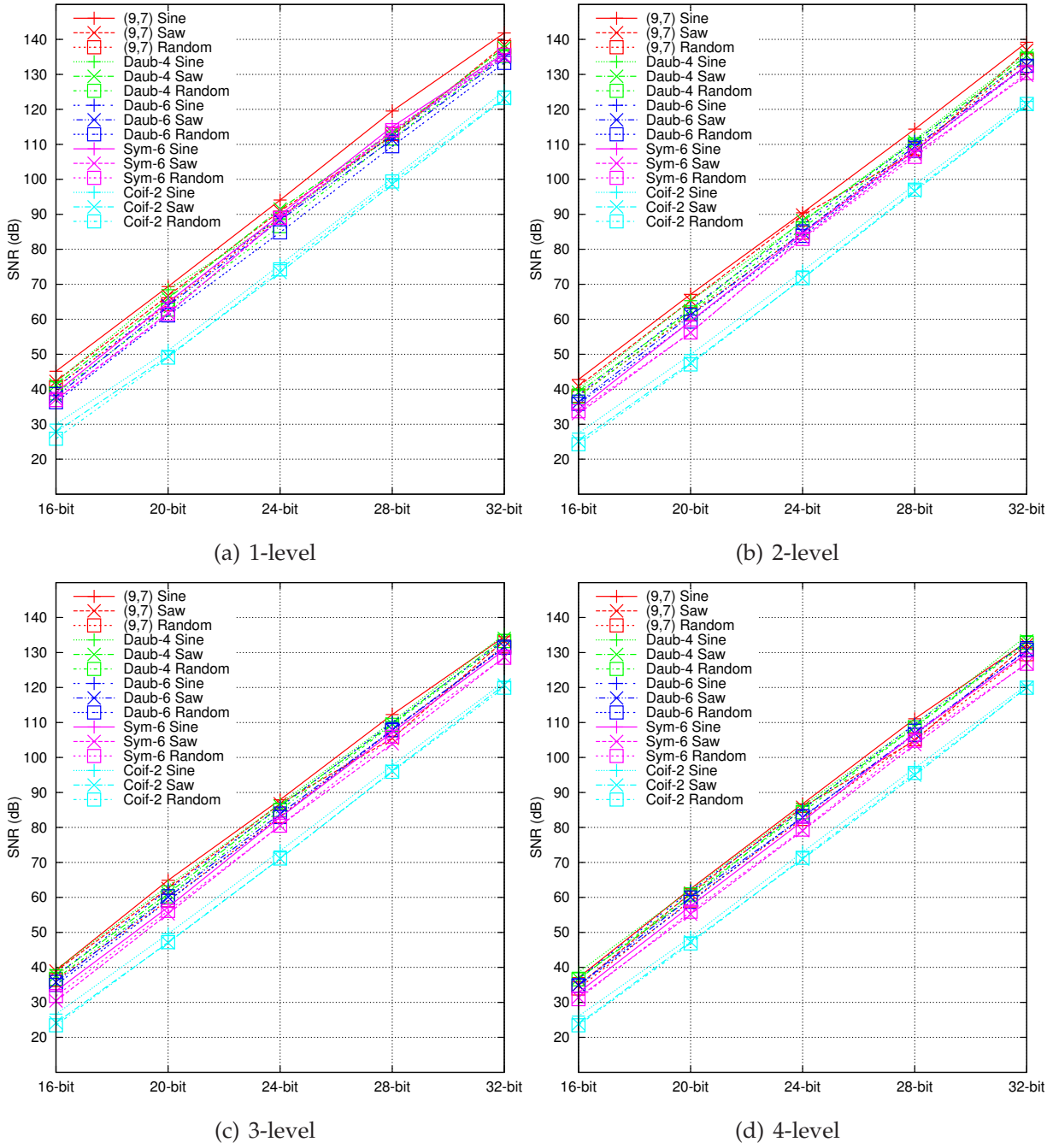
Because wavelet transform is a multiresolution signal processing tool, we perform various levels of DWT and DWP decomposition and reconstruction to give a better overview of the performance of our wavelet processors. The SNRs of the different data width implementations for different levels of DWTs and DWPs are reported in **Fig**. 5.7 and **Fig**. 5.8 for wavelet processors with fixed-point arithmetics. For wavelet processors with floating-point arithmetics, they are reported in **Fig**. 5.9 and **Fig**. 5.10.

Let us have a detailed analysis on the DWT results of the fixed-point wavelet processors first, as depicted in **Fig**. 5.7. For 16-bit implementation, the SNRs vary between 26 dB and 45 dB depending on the signals and the wavelet filters used for the 1-level DWTs. As we expect, because of the slightly higher dynamic coefficient range in Coiflet-2 wavelet filter, the resulting SNRs of DWT with this filter are around 10 dB lower compared to SNRs with Daubechies wavelet filters. The SNRs decrease when more transform levels are involved. 4-level decomposition lower the achievable SNRs between 3 dB to 10 dB. It is due to the fact that the resulting low-pass component, which degrades the signal quality, is transformed again. The SNRs increase linearly with the increasing of data width because we apply proportional shifting on the corresponding data width realizations. For 32-bit implementations, the SNRs vary between 123 dB and 142 dB for the 1-level DWTs, depending again on the filters and the signals, and between 120 dB and 135 dB for the 4-level DWTs. DWP decomposes all frequency components. Therefore it has direct impact on the SNRs, especially when higher transform levels are exercised. As depicted in **Fig**. 5.8, around 5 dB decrease in the achievable SNRs can be expected for 4-level DWPs compared to the DWTs.

Note that we implement a fixed shifter that is scaled proportionally with the increase of the data width. The shift amounts are optimized for samples with 8-bit data width.

Tab. 5.17: Decomposition lifting coefficients of (9,7), Daub-4, Daub-6, Symlet-6, and Coiflet-2 wavelet filters.

Type	(9,7)		Daub-4		Daub-6		Symlet-6		Coiflet-2	
Predictor	-1.500 z^0	-1.500 z^1								
Updater	0.063 z^{-1}	0.063 z^0	1.732 z^0		2.425 z^0		-0.227 z^0		-2.530 z^0	
Predictor	0.800 z^0	0.800 z^1	0.067 z^{-1}	-0.433 z^0	0.079 z^{-1}	-0.352 z^0	-1.267 z^{-1}	0.216 z^0	-0.240 z^{-1}	0.342 z^0
Updater	0.938 z^{-1}	0.938 z^0	-1.000 z^1		-2.895 z^1	0.561 z^2	0.505 z^1	-4.255 z^2	3.163 z^1	15.268 z^2
Predictor					-0.020 z^{-2}		0.045 z^{-3}	0.233 z^{-2}	0.006 z^{-3}	-0.065 z^{-2}
Updater							-18.389 z^3	6.624 z^4	-63.951 z^3	13.591 z^4
Predictor							0.144 z^{-5}	-0.057 z^{-4}	0.001 z^{-5}	0.002 z^{-4}
Updater							-5.512 z^5		-3.793 z^5	
Normalizer	1.131	0.884	0.517	1.932	0.432	2.315	-0.599	-1.671	0.108	9.288

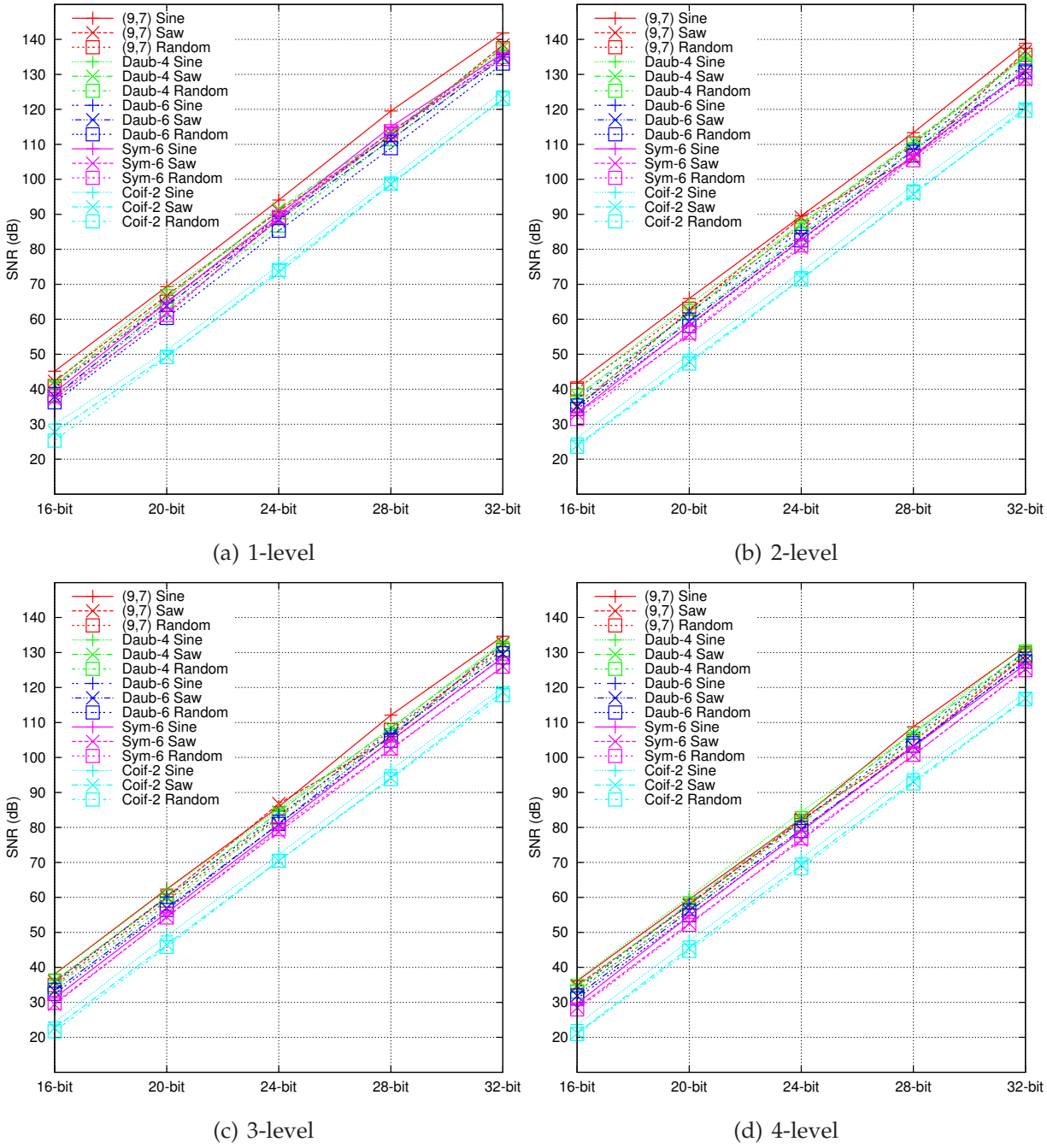


(a) 1-level (b) 2-level

(c) 3-level (d) 4-level

Fig. 5.7: Multilevel DWT using fixed-point wavelet processors with different data width implementations.

Therefore, even if we increase the processor data width, it does not necessary mean that the sample range also increases. It only tells us that the computations are performed with the selected data widths to deliver higher precision. This means that if samples with higher data width are used, we need to modify the shift amount and/or to rescale the samples in order to deal with the arithmetic overflow. Therefore, the 32-bit realizations of our fixed-point wavelet processors deliver higher SNRs, but they suffer from the limited input range, in this case 8-bit. Additionally, fixed-point implementations are also



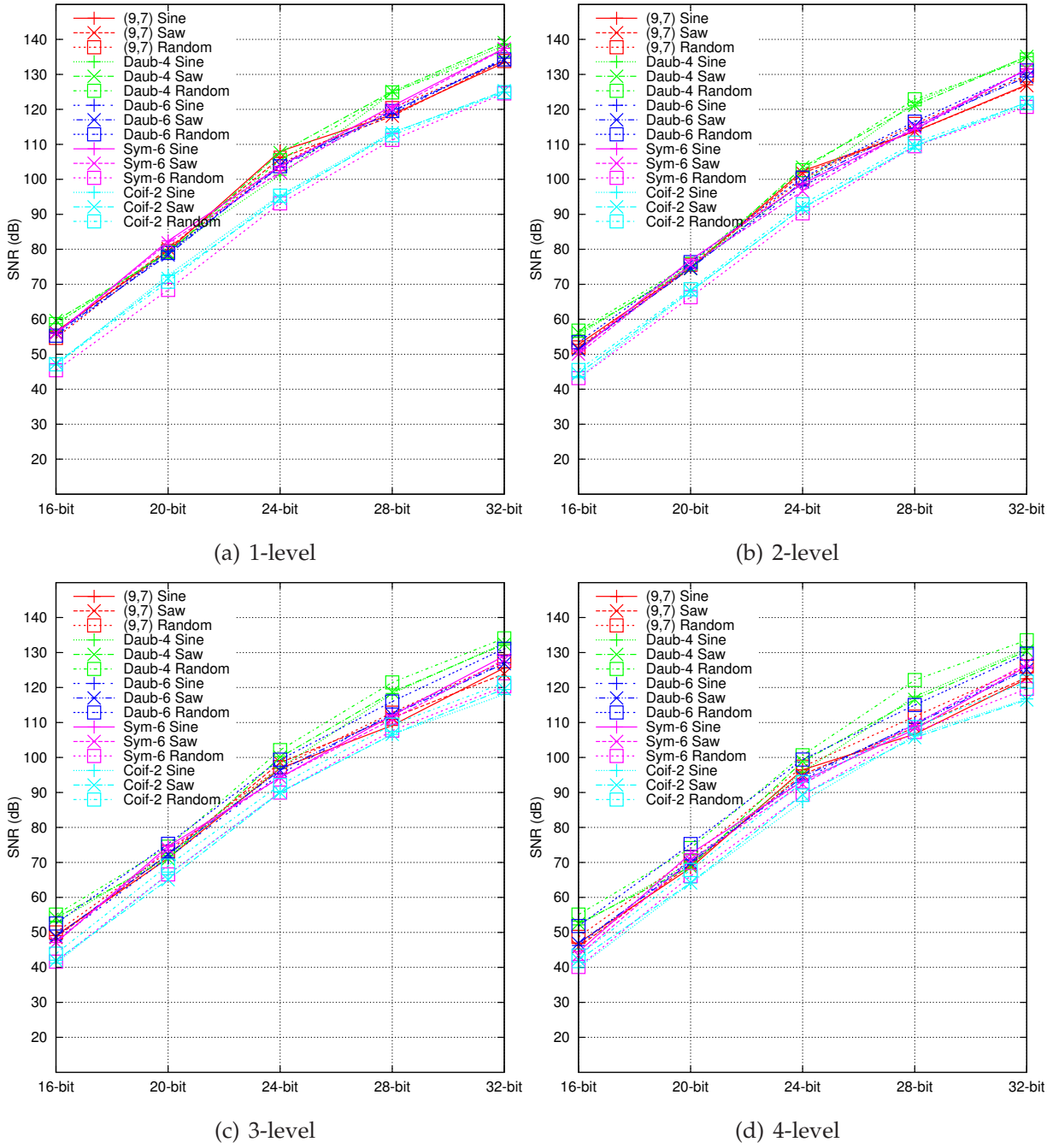
(a) 1-level (b) 2-level

(c) 3-level (d) 4-level

Fig. 5.8: Multilevel DWP using fixed-point wavelet processors with different data width implementations.

sensitive to the lifting coefficients. If the range of the lifting coefficients used during the transform is wider, we might need to adjust the shift amount to adapt with higher dynamic values in the temporary transform results, and sacrifice the precision as the side effect. Lastly, performing higher level transforms also leads to higher dynamic values in the final transform results. Therefore, the decision of the shift amount is an important issue during the design phase of the fixed-point wavelet processors.

On the other side, the floating-point wavelet processors deliver greater flexibility and

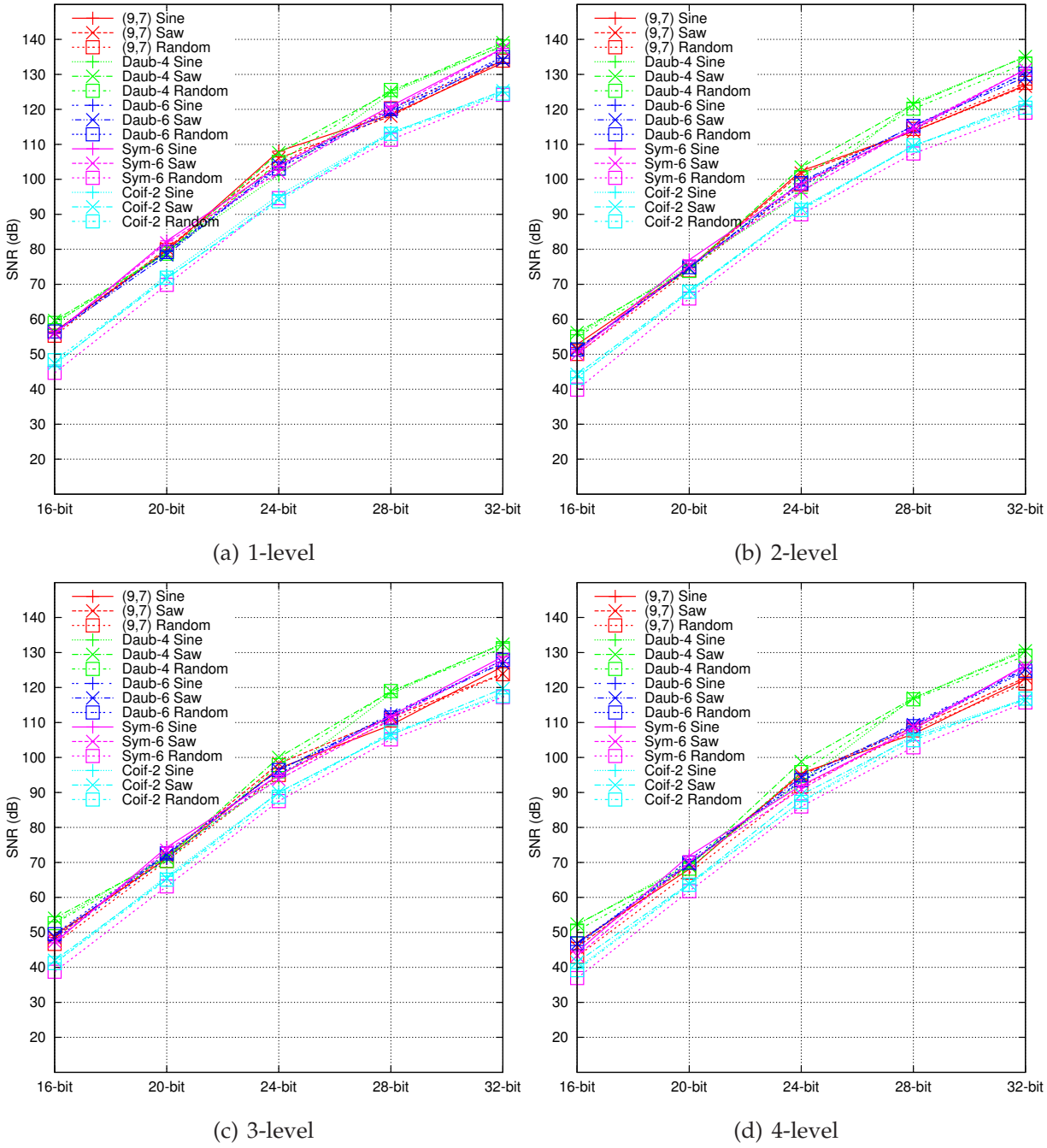


(a) 1-level (b) 2-level

(c) 3-level (d) 4-level

Fig. 5.9: Multilevel DWT using floating-point wavelet processors with different data width implementations.

higher dynamic range, by still maintaining the small residues of the final transform results, especially on the high-pass components. The DWT results of several wavelet filters with floating-point wavelet processors are depicted in **Fig**. 5.9. For 16-bit implementations, SNRs vary between 45 dB and 60 dB for the 1-level DWT, that is around 20 dB higher compared to the same implementations with fixed-point arithmetics. The reason is obvious. In the fixed-point realizations, we need to define at the design time the shift amount in a certain way, so that all possible test input signals can be properly transformed with-



(a) 1-level (b) 2-level

(c) 3-level (d) 4-level

Fig. 5.10: Multilevel DWP using floating-point wavelet processors with different data width implementations.

out introducing overflow for various wavelet filters and with different levels. Therefore, in order to avoid overflow in the arithmetics when higher transform levels are computed, we need to set the shift amount slightly lower. This has impact on the transforms of the lower levels by reducing the number of bits used for precision. This issue is relaxed in the floating-point arithmetics. The resulting low-pass components have higher dynamic range whilst the high-pass components comprise mostly of lower dynamic range compared to the low-pass ones. All of them can be efficiently computed and stored in the

floating-point format, even with the smaller number of bits used to store the mantissa.

Multilevel transforms do indeed have impact not only on fixed-point wavelet processor, but also on floating-point wavelet processor. The achievable SNRs decrease along with the increase of the transform level. For 4-level DWT, the 16-bit floating-point implementations achieve SNRs between 40 dB and 55 dB, which is still 10 dB to 20 dB higher compared to the fixed-point wavelet processor with the same data width.

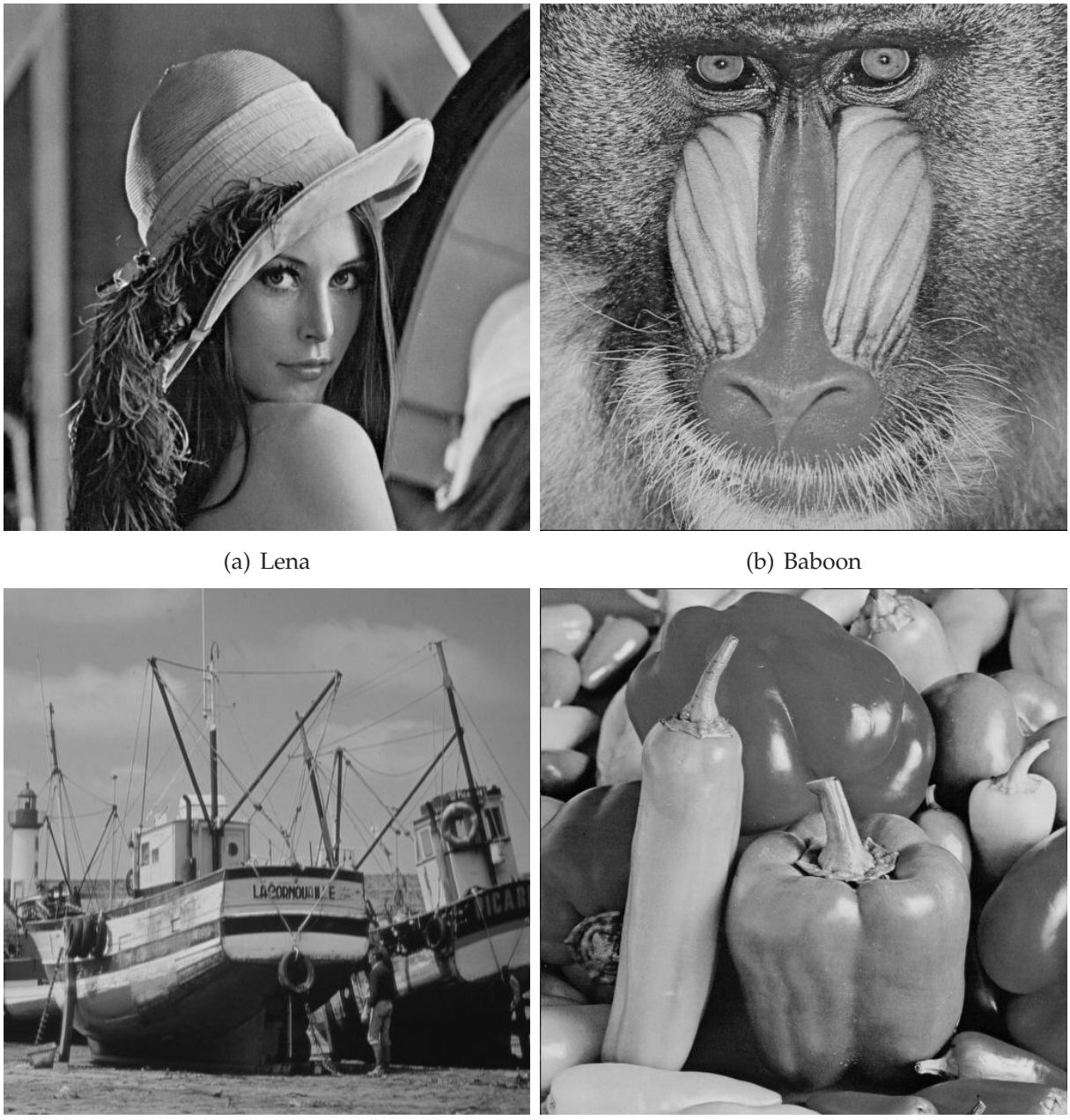
Similar to the fixed-point architectures, by increasing the data width, the achievable SNRs also increase. The increases are almost linear due to the fact that the number of bits spent for mantissa does not proportionally grow along with the data width. Therefore, the achievable SNRs do not scale linearly with the increase of data width. Nevertheless, the number of bits spent for exponent grows with the increase of data width. It means that the floating-point architectures with higher data widths have higher computation range too. This is not the case in the fixed-point architectures. For 32-bit realizations, the achievable SNRs for 4-level DWTs are between 117 dB and 134 dB, and for 4-level DWPs are between 116 dB and 131 dB. This indicates that floating-point architectures are less sensitive for DWPs compared to fixed-point ones.

5.6.2 Performances on 2D Signals

In addition to the analysis on performance with three different synthetic 1D input samples, we also provide SNR for 2D natural signals. Four different image sources are taken as references, shown in **Fig**. 5.11. These images have 8-bit greyscale resolution with the size of 512×512 pixels. They are transformed using the same four types of wavelet filters used in the previous SNR performance analysis on 1D signals. We perform 3-level DWT and DWP of these images. The same cases as in the 1D analysis, the SNRs are computed from the reconstructed signals, that is the cumulative SNRs from decomposition and reconstruction. These SNRs are plotted in **Fig**. 5.12.

As we expected, multidimensional transforms lead to lower achievable SNRs compared to 1D transforms. This can be explained as follows. After the first row transforms, the images are split into two components, i.e. L and H. These components are further transformed column-wise, which generate LL, LH, HL, HH components. Note that the low-pass component L is transformed into LL and LH while the high-pass component H, which contains the details of the image, is transformed into HL and HH. Because H comprises of the details of the image in one direction, most of the coefficients of this component have smaller values. Therefore, transforming this high-pass component contributes to a lower SNR. In 3-level 2D DWPs, the signal quality degradation during the transforms is more dominant. This is because not only LL component is further transformed, but also LH, HL, HH components, which contain the image details, are involved during the transforms in order to generate 64 tiles with equal size.

Similar to the 1D transforms, the fixed-point architectures achieve lower SNRs com-

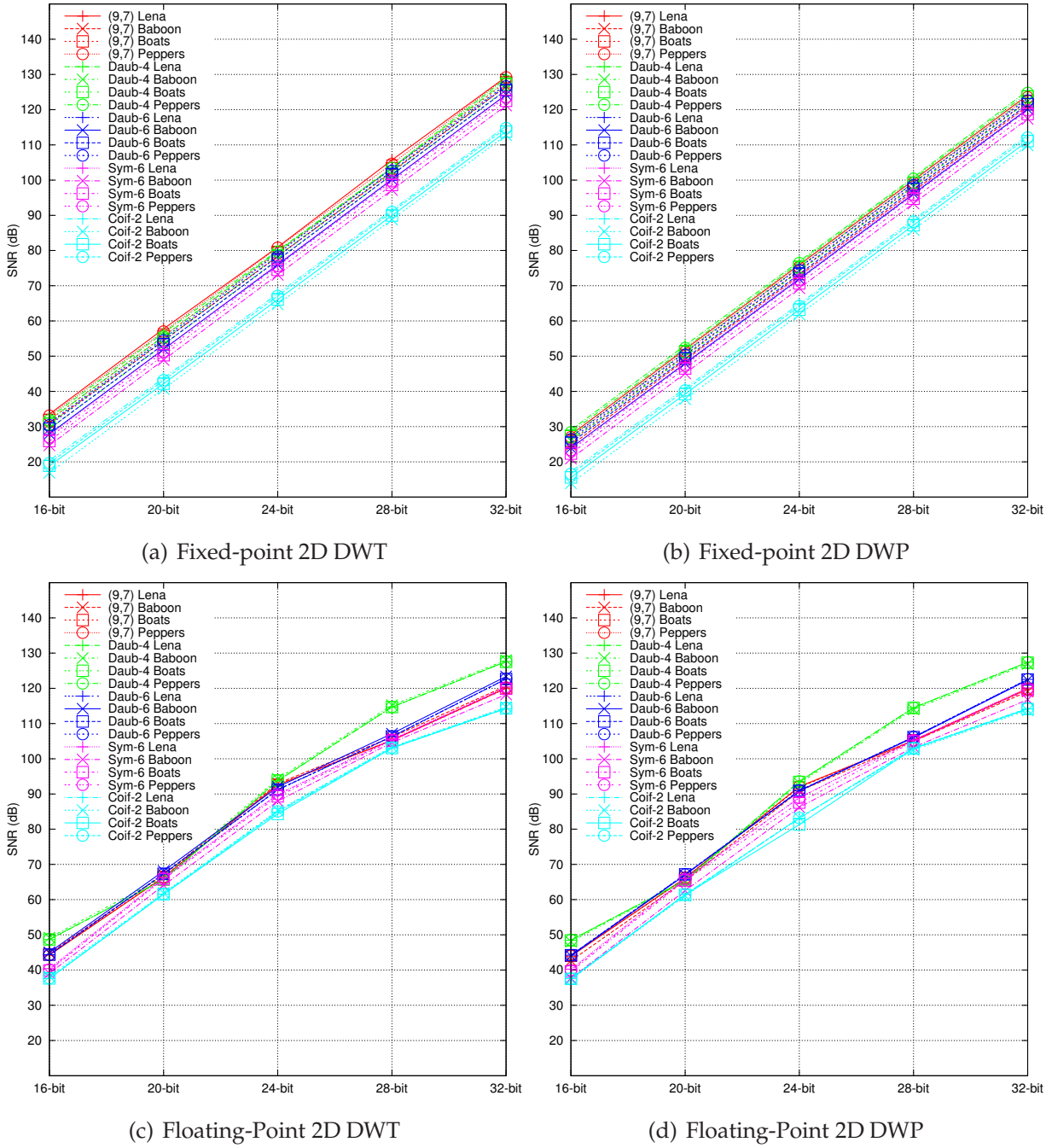


(a) Lena (b) Baboon

(c) Boats (d) Peppers

Fig. 5.11: 2D image sources.

pared to the floating-point architectures. For 16-bit implementation, the SNRs vary between 17 dB and 34 dB for DWTs and between 14 dB and 30 dB for DWPs, depending on the image source and on the wavelet filter. The use of Coiflet-2 wavelet filter contributes to a lower SNRs due to the higher lifting coefficient range. The (9,7) filter on the other side delivers adequate SNRs between 31 dB and 34 dB for DWTs and between 25 dB and 28 dB for DWPs. The SNRs increase linearly with the increase of data width. The 32-bit implementation achieves SNRs between 113 dB and 129 dB for DWTs and between 110 dB and 125 dB for DWPs.



(a) Fixed-point 2D DWT

(b) Fixed-point 2D DWP

(c) Floating-Point 2D DWT

(d) Floating-Point 2D DWP

Fig. 5.12: SNR values of various 2D images.

The floating-point implementations in the other hand are less sensitive to the SNR degradation in multidimensional transforms. For 16-bit implementation, the SNRs vary between 38 dB and 49 dB for DWTs and between 37 dB and 49 dB for DWPs. Interesting to notice that the achievable SNRs for DWT and DWP do not differ significantly in the floating-point architectures. Whilst the major contribution to the lower SNRs is the same as the fixed-point implementations, i.e. the use of Coiflet-2 wavelet filter, the upper bound of the SNRs in the floating-point implementations are given by the Daub-4 wavelet filter. The 32-bit single-precision floating-point realizations achieve SNRs between 114 dB and

128 dB for DWTs and between 114 dB and 127 dB for DWPs.

5.7 Benchmarks

We develop four different types of wavelet processors, i.e. two different architectures for fixed-point wavelet processors, i.e. resource-aware and high-performance architectures, and their corresponding floating-point architectures. All of them utilize a pipeline mechanism so that the samples can be fed and computed every two clock cycles in case of resource-aware fixed-point and resource-aware floating-point wavelet processor, and every one clock cycle in case of high-performance fixed-point and floating-point wavelet processor. Because different number of pipelines is used for different architectures, the computation performance of the wavelet processors also differs between implementations.

In resource-aware fixed-point wavelet processor, the total latency on each PE is 4 clock cycles. One clock cycle is consumed by the input registers, 1+1 by the multiplier (two multiplications are performed), and 1+0 by the adder (two summations are performed where one cycle is *stolen* from the multiplier). Additional sample latency (2 clock cycles per future sample) will add-up to the total latency on the PEs which require this feature. The PE that is configured to perform the normalization step has latency of 3 clock cycles. In high-performance fixed-point wavelet processor, the total latency on each PE is also the same, i.e. 4 clock cycles. One clock cycle is consumed by the input registers, one by the multipliers, and two by the adders. Because two multipliers are used, the PE that is configured as normalizer requires only two clock cycles. Further, one clock cycle per future sample will add-up to the total latency on the PEs which use future samples.

Floating-point realizations add more pipelines in order to boost the operating speed. In resource-aware floating-point wavelet processor, the total latency on each PE is 7 clock cycles. One clock cycle is used by the input registers, 2+1 by the multipliers, and 3+0 by the adders (one cycle is shared with the multiplier). Similar to the fixed-point architecture of the same type, two clock cycles per future sample will add-up to the total latency. The PE configured as normalizer requires 4 clock cycles. We further optimize our floating-point adder with three inputs to adapt with the high-performance architecture. Therefore, the total latency on each high-performance floating-point PE is 6 clock cycles. One is used by input registers, two by the multipliers, and three by the adder. The normalizer of this type of PE requires only 3 clock cycles.

The computation performance of our wavelet processors does not only depend on the type of the processor, but also depends on the type of the transform (i.e. DWT or DWP), the number of transform levels, type of wavelet filters, etc. In order to give more objective evaluations regarding of the speed of our wavelet processors, we detail the time needed to compute the transform of (5,3) wavelet filter with different levels and different types of transforms. Because our wavelet processors can be designed to have arbitrary data

Tab. 5.18: Benchmarks for 1D signal with (5,3) filter.

Transform	Fixed-Point				Floating-Point			
	Resource-Aware		High-Performance		Resource-Aware		High-Performance	
	DWT	DWP	DWT	DWP	DWT	DWP	DWT	DWP
1-level Forward	533	533	274	274	541	541	279	279
1-level Inverse	533	533	274	274	541	541	279	279
2-level Forward	1067	1072	549	554	1083	1088	559	564
2-level Inverse	1067	1072	549	554	1083	1088	559	564
3-level Forward	1086	1627	565	850	1110	1651	580	865
3-level Inverse	1086	1627	565	850	1110	1651	580	865
4-level Forward	1364	2214	712	1178	1396	2246	732	1198
4-level Inverse	1364	2214	712	1178	1396	2246	732	1198
5-level Forward	1255	2865	664	1570	1295	2905	689	1595
5-level Inverse	1255	2865	664	1570	1295	2905	689	1595
6-level Forward	1469	3644	779	2090	1517	3674	809	2108
6-level Inverse	1469	3644	779	2090	1517	3634	809	2108

width, which achieve different operating frequencies, the time needed to compute each transform is measured in terms of the number of clock cycles. **Tab**. 5.18 summarizes the results. 512 samples are used as a reference. In all cases, inverse transforms (reconstructions) consume the same amount of clock cycles as forward transforms (decompositions) on the same type of wavelet processor. This is obvious because the polyphase representation of the inverse transform of (5,3) filter is obtained directly by negating the filter coefficients and by reversing the lifting order. The high-performance wavelet processors of both types deliver higher throughput because they process two samples every clock cycle, which leads to almost two folds time reduction. It is not possible to reduce the number of clock cycles on the high-performance wavelet processors by half because of the computation latencies introduced by each PE. As floating-point PEs introduce more pipelines, both floating-point wavelet processors require more clock cycles compared to the fixed-point wavelet processors of both resource-aware and high-performance architectures. The same benchmarks for (9,7) wavelet filters are summarized in **Tab**. 5.19.

The time needed to compute higher levels of forward DWT and inverse DWT does not always increase proportionally. If we look at the 2-level and 3-level DWT transforms, they only differ slightly. Furthermore, 5-level DWT requires less time compared to 4-level DWT. Recall our discussion on transform processes in **Sec**. 4.8. We perform selective copy to synchronize the content of both banks. When 2-level DWT decomposition is executed, the low-pass component L_1 in the shadow bank is decomposed into L_2 and H_2. These two resulting frequency components are stored in the primary bank, which is the bank to store the final transform results. The high-pass component H_1 is however stored in the shadow

Tab. 5.19: Benchmarks for 1D signal with (9,7) filter.

Transform	Fixed-Point				Floating-Point			
	Resource-Aware		High-Performance		Resource-Aware		High-Performance	
	DWT	DWP	DWT	DWP	DWT	DWP	DWT	DWP
1-level Forward	548	548	286	286	568	568	298	298
1-level Inverse	548	548	286	286	568	568	298	298
2-level Forward	1097	1104	573	582	1137	1146	597	606
2-level Inverse	1097	1104	573	582	1137	1146	597	606
3-level Forward	1131	1680	601	902	1191	1748	637	938
3-level Inverse	1131	1680	601	902	1191	1748	637	938
4-level Forward	1424	2296	760	1270	1504	2398	808	1318
4-level Inverse	1424	2296	760	1270	1504	2398	808	1318
5-level Forward	1330	2992	724	1734	1430	3103	784	1767
5-level Inverse	1330	2992	724	1734	1430	3103	784	1767
6-level Forward	1559	3825	851	2119	1679	4000	923	2152
6-level Inverse	1559	3825	851	2119	1679	4000	923	2152

bank. Therefore, we need to copy the content of H_1 to the primary bank. In case of 3-level DWT decomposition, we do not need to copy the content of H_1 from the shadow bank to the primary bank, because the shadow bank is the final bank to store the transform results. Still, we need to copy the content of H_2 from the primary bank to the shadow bank. The same principles are applied to optimize the 5-level DWT decomposition, and so on.

DWP decomposition and reconstruction are more computational intensive because all frequency components are involved. Additionally, no selective copy which decreases the computation time is executed in case of DWP transforms. Therefore the time needed to compute higher level of DWT decomposition and reconstructions increases almost linearly with the increase of number of level.

Note that the results in **Tab**. 5.18 are the time needed by the processor to compute the transform. They do not include the data preparation time. In case of resource-aware processors, additional 512 clock cycles are needed to prepare the samples, if we want to compute the transform of 512 samples. In case of high-performance processors, two samples can be fed every clock cycle. Thus, we only need 256 additional clock cycles. During retrieving, because our memory banks are exclusive, read and write operations can be executed at the same time. Therefore, while we prepare the next samples on one bank, we can also read the resulting transform from the other bank. More details about the process can be seen in **Sec**. 4.7.3.

Typical performance benchmark of a wavelet processor is by taking JPEG2000 as a case

Tab. 5.20: Benchmarks for JPEG2000 with (5,3) filter.

Type	Fixed-Point		Floating-Point	
	Resource-Aware	High-Performance	Resource-Aware	High-Performance
Clock Cycles	1413888	720384	1428224	729344
16-bit	225 fps	477 fps	318 fps	623 fps
32-bit	172 fps	359 fps	211 fps	409 fps

Tab. 5.21: Benchmarks for JPEG2000 with (9,7) filter.

Type	Fixed-Point		Floating-Point	
	Resource-Aware	High-Performance	Resource-Aware	High-Performance
Clock Cycles	1440768	741888	1476608	763392
16-bit	221 fps	463 fps	307 fps	595 fps
32-bit	169 fps	349 fps	204 fps	391 fps

study. Because JPEG2000 deals with 2D transform, we need to compute row transforms first, followed by column transforms. All transforms are iterative 1D transforms on both dimensions. In other words, we cannot directly compute 3-level DWT decomposition of row images, followed by 3-level DWT decomposition of column images. The data preparation will be the main contribution because of this scheme because we need to feed all row and column images several times.

Let us first have detail observations on resource-aware fixed-point wavelet processor and the time required to compute 512×512 8-bit greyscale image. From **Tab**. 5.18, 533 clock cycles are needed to compute 1D transform. This leads to 21 clock cycles computation latency which is independent from the signal length. For one row image, we need 512 + 533 = 1045 clock cycles. Because we have 512 row images, the total time to compute row transforms is $512 \times 1045 = 535040$ clock cycles. After all row images are transformed, we need to transform the column images, which is also 535040 clock cycles. Thus, 1-level DWT decomposition of 512×512 image requires 2×535040 = 1070080 clock cycles. The next 2D transform involves only a quarter of the size of the original image, i.e. 256×256 image. With the same computation latency time, the time needed to compute all row images is $(256 + (256 + 21)) \times 256 = 136448$ clock cycles, which leads to $2 \times 136448 = 272896$ clock cycles for the whole row and column images. The last 2D DWT decomposition involves 128×128 image, which requires $2 \times (128 + (128 + 21)) \times 128 = 70912$ clock cycles. The total time required to compute 3-level 2D DWT decomposition of 512×512 image is 1413888 clock cycles. For 16-bit realization, it corresponds to $(1413888/319.49)\mu s = 4425.45\mu s$, and for 32-bit, it corresponds to $(1413888/244.50)\mu s = 5782.77\mu s$. Therefore, the resource-aware fixed-point wavelet processors can process as much as 225 frame per second (fps) for 16-bit and 172 fps for 32-bit. The benchmarks of the rest of the architectures for both

(5,3) and (9,7) wavelet filters are summarized in **Tab**. 5.20 and **Tab**. 5.21. The amount of the clock cycles needed along with their achievable fps for 16-bit and-32-bit realizations are shown.

5.8 Comparisons

There exist numbers of architectures to compute wavelet transforms based on their lifting scheme representations. Most of the architectures are dedicated to compute DWT with (5,3) and (9,7) wavelet filters used in JPEG2000. Therefore, those architectures are much simpler. Because they are designed to cope with these two wavelet filters and the input samples are restricted to have 8-bit width, dynamic fluctuations on the temporary and the final transform results are very small. Additionally, JPEG2000 requires only 3-level DWT transform on a 2D image. Taking all of these into account makes fixed-point arithmetics a good candidate to be used as the computation core.

Most of the developed architectures implement various target technologies and fabrication processes with specific features and development goals. Thus, it is not easy to compare them by the same criteria. Nevertheless, we try to compare the performance of our wavelet processor with the other existing architectures. **Tab**. 5.22 summarizes the comparison. Architecture from Andra *et al.* [?] uses small amount of calculation resources, but it uses large amount of memory. Additionally, because (9,7) filter is split into two stages, which are scheduled for row and column processors, the control scheme of this architecture is very complex. The sample processing speed of this architecture is only $f/2$. It means that it can only compute one sample per clock cycle. Architecture from Dillen *et al.* [?] uses FPGA as a target device. Therefore, in order to minimize the wiring cost and also to cope with dual-port memory in FPGAs, its processing speed is also $f/2$. Architecture from Seo *et al.* [?] is synthesized with 0.35-μm process even it is the latest development among other architectures. The processing speed of this architecture is f, which delivers image processing rate of 219 fps. All of these three architectures support only 3-level 2D transforms with (5,3) and (9,7) wavelet filters. The last architecture which does not deal with JPEG2000 is developed by Wang *et al.* [?]. The DDC architecture can only compute DWP with Daub-4 wavelet filter and has only 50 % utilization. If different filters are used, the architecture and its interconnects need to be redesign. Because of the constrained butterfly operations, different wavelet filters cannot be easily used, even if they have the same lifting length as Daub-4. Additionally, DDC architecture exploits zero-padding in order to compute the transform in the boundary regions. This approach increases the sample size during each transform in order to make it reversible. If we apply this rule to the DWP where all existing bands need to be decomposed, the resulting size of the final transform will increase significantly. Finally, because of lack of the memory, DDC architecture only considers data stream as inputs. It does not have any memory interface to control how the samples are fed and stored, which are the main issue in DWP. Therefore, although DDC architecture is capable of computing DWP, basically, it is only a

Tab. 5.22: Comparison with other lifting-based architectures.

Architecture	Freq.	Area	Filter	Transform	D. Width	Mem. Size	Arithmetic	Speed
Andra [?]	200 MHz (0.18-μm)	2.8 mm^2	(5,3) (9,7)	DWT, IDWT	16-bit	128	Fixed-Point	$f/2$
Dillen [?]	110 MHz (FPGA)	–	(5,3) (9,7)	DWT, IDWT	16-bit	256	Fixed-Point	$f/2$
Seo [?]	150 MHz (0.35-μm)	5.6 mm^2	(5,3) (9,7)	DWT, IDWT	12-bit	512	Fixed-Point	f
Wang [?]	100 MHz (0.18-μm)	1.1 mm^2	Daub-4	DWP	18-bit	Data Stream	Fixed-Point	$f/2$
Ours	245 MHz	3.4 mm^2	Arbitrary	DWT, IDWT, DWP, IDWP	32-bit	512	Fixed-Point	$f/2$
	259 MHz	4.8 mm^2						f
	355 MHz	3.7 mm^2					Float.-Point	$f/2$
	299 MHz	4.9 mm^2						f

group of PEs for computing DWTs.

5.9 Concluding Remarks

Our wavelet processors can compute not only DWT decomposition and reconstruction with various wavelet filters, but also DWP decomposition and reconstruction. Compared to the other architectures, our processors can be used to compute transforms on 1D signal as well as N-dimensional signal, for both DWT and DWP. Additionally, we do not limit the level of the transform that can be carried out by our wavelet processors. Our wavelet processors have higher throughput, even for the resource-aware processors which have processing speed of $f/2$. Using lossless JPEG2000 configuration, the processor can transform 512×512 image with 225 fps for 16-bit resource-aware fixed-point implementation, and increase the throughput at least two folds when the high-performance versions are taken. For 32-bit architecture of the same type, a throughput of 172 fps can be achieved by the processor. With slightly increase in chip area, floating-point wavelet processors achieve higher SNRs compared to the fixed-point implementations. Whilst we need to determine the shift amount during the design time in the fixed-point implementations to achieve a good SNR performance while still delivering the capability to exercise different wavelet filters, we do not have such problem in the floating-point implementations. Additionally, the input sample range in the floating-point wavelet processors increase along with the increase of the data width, whilst this is not the case in the fixed-point wavelet processors.

Chapter 6

Conclusions and Future Works

Contents

6.1 Contributions of the Work 160

6.2 Directions for Future Work 161

Traditional wavelet transforms use filter banks to decompose the signal into different frequency bands. The advantages of the filter-based implementation are that the architecture is straightforward and changing the wavelet kernel does not alter the architecture so much, because in most architectures, we only need to change the wavelet coefficients and adapt the length of the wavelet filter during the design time. Although the realizations are straightforward, filter banks offer less flexibility. If another class of filters with higher order is used, it is necessary to change the design of the filter banks to cope with longer convolution. Contrary to the filter banks approach, which separates the signal into low and high frequency components and performs the decimation on both signals afterwards, the second generation of wavelets, more popular under the name of lifting scheme, reduces the computation by performing the decimation in advance. Lifting-based architectures become very popular because of the efficiency these architectures deliver. As the decimation is performed in advance in the lifting scheme, it can be expected that lifting-based DWT architectures are twice efficient compared to the filter-based when dyadic DWT is concerned.

In the recent years, wavelets have become a hot topic in both industry and research fields due to the adoption of the wavelet transforms by the JPEG committee in their new JPEG2000 standard. In the transform block of JPEG2000, two different wavelet filters can be applied depending on the compression methods. For lossless compression, (5,3) filter is used because it has integer coefficients and requires no scaling, whereas for lossy compression, (9,7) filter is used because it can exploit the locality property of the signal better compared to (5,3) filter. In addition to the block transform of JPEG2000, wavelets are also applied in many other applications such as feature extraction, feature detection, voice synthesis, etc.

6.1 Contributions of the Work

To cope with different application demands, we have developed hardware architectures to compute wavelet transforms based on their lifting scheme representations in a more generalized manner. Our wavelet processors are divided into two categories based on their arithmetic cores, i.e. fixed-point and floating-point architectures. On each category, two types of architectures are developed, i.e. resource-aware wavelet processor that handles with lower throughput and high-performance wavelet processor that increases the performance two folds. In total, four different wavelet processors are developed. The processors are based on cross-chained PEs to compute prediction and update atom functions of the lifting steps. All architectures offer higher flexibilities in terms of data width, memory size, memory organization, number of PEs, context size, etc.

There exist two different types of wavelet transforms, i.e. DWT and DWP. In DWTs, only the resulting low-pass component is used as an input for the next transform level, whereas in DWPs, both low-pass and high-pass components are further decomposed. Our wavelet processor can compute DWT decomposition and reconstruction, as well as DWP decomposition and reconstruction. In addition to the 3-level DWTs on JPEG2000 which are purely 2D DWTs and they can be decomposed easily into 1-level 1D DWT on each step, our wavelet processors support also N-level DWT decomposition and reconstruction of 1D signal and N-level DWP decomposition and reconstruction of 1D signal. Higher dimensions signal can easily be transformed using both DWT and DWP methods by treating the signal independently in each dimension. As an example, 2D DWT for JPEG2000 comprises of two independent transforms, i.e. row images followed by column images transform. These kinds of transforms, along with multilevel 1D transforms, are supported by our wavelet processors.

We have also developed an efficient algorithm to factorize the wavelet filter coefficients into their polyphase representations. By matching two terms of the Laurent polynomials, the polyphase matrix can be decomposed into several polyphase matrices with smaller prediction and update functions. This leads to an efficient realization that can be mapped into hardware and used to compute both DWTs and DWPs with different wavelet filters, by maintaining the diversities of the wavelet filters that can be used as the wavelet kernel.

To deliver greater dynamic computation range, we have also designed wavelet processors that are based on floating-point arithmetics. The need of floating-point grows, especially when longer wavelet filters are used and higher transform levels are performed. Because factoring different wavelet filters lead to different polynomial results on the prediction and update functions of the polyphase matrix, the prediction and update coefficients as the results of the polynomial factorization may contain higher dynamic range. The floating-point numbers are less sensitive to this issue, which make them suitable for general purpose wavelet transforms that use different kinds of wavelet filters. Because two matching terms are used, we require special floating-point adder that can accept

three inputs to compute prediction and update functions. The major challenge in designing the floating-point wavelet processors is the design of this adder. In order to reduce the latency time introduced by the adder, we have developed two 3-input floating-point adders; one is optimized for our resource-aware wavelet processor and the other one is optimized for our high-performance wavelet processor.

Because we based our design on the pipeline mechanism, our wavelet processors achieve higher operating frequency. Additionally, they have high throughput. For 16-bit resource-aware wavelet processors, 225 fps can be processed by the fixed-point architecture and 318 fps by the floating-point architecture. The high-performance wavelet processors increase the performance around two folds. For 16-bit high-performance wavelet processors, 477 fps can be achieved by fixed-point architecture and 623 fps by floating-point architecture. The benchmarks are measured by using (5,3) wavelet filter in JPEG2000 and by taking into account the image with the size of 512×512 pixels.

6.2 Directions for Future Work

We have developed an algorithm to map the wavelet filters into the lifting scheme representations. The algorithm is based on finding Eucledian distance by using the greatest common divisor. Our algorithm relaxes the matching of two terms of the Laurent polynomials by searching also the next possible and better solutions. The optimization cost of our algorithm is based on the minimum possible dynamic range of the polynomial residue. We do not take into account the resulting quotient during the choosing of possible solutions. Therefore, even the algorithm provides a better solution compared to matching two directly neighbouring terms, it is still less optimal. It may lead to larger dynamic coefficients range on the final matching. Additionally, our algorithm tries to find a best solution from the lowest polynomial degree toward the highest polynomial degree. There exist different algorithms to find greatest common divisor between two polynomials. To name one of them, solution to give a better polynomial factorization can be computed by alternating between the matching of the lowest and matching of the highest polynomials. We need to consider better optimization costs in respect with hardware constraints. Thus, signal degradation during the transforms can be minimized.

In order to reduce the data preparation time, we have also developed a solution by adding extra memory bank into our wavelet processors. While one bank is used to compute the transform, the next samples can be prepared on the other bank. This solution works not only for N-dimensional signals, but also on 1D signals that exercise multilevel transforms. Other than minimizing the data preparation time, this additional bank has no other use. Wavelet transforms work with larger number of samples. One idea is to use this additional bank along with the existing bank, and treat them as one single bank. Using this approach, the wavelet processors can accept non-segmented input data samples two times larger without increasing the size of the memory.

Computing wavelet transforms on the boundary regions requires special treatment. There exist at least three different solutions to cope with this issue. The first solution is by adding zero padding on the signal. This approach increases the sample size during each transform. The second solution, which is used by our wavelet processors, is by exploiting the signal periodicity. This solutions works for all types of wavelet filters and all types of signal. The third solution is by exploiting the mirroring extension of the signal. This solution works only for symmetric wavelet filters. On some applications, mirroring extension might be required. Another boundary treatment is done by exercising other types of wavelet filters that use less or no signal extension, such as Haar wavelet. This requires special treatment by gradually change the types of the filters and can be applied on selective wavelet filters in order have a perfect reconstruction.

We consider only full-tree decomposition and reconstruction in DWPs. However, for some applications, we might want to select and keep one or several frequency bands from the previous decomposition levels. This scheme is called basis selection. Selecting the best basis is another challenging task and it depends on applications. As an example, signal compression prefers to have basis with more zero signals.

Appendix A

Fourier Analysis

Fourier transform is one of the best known of the integral transforms and one of the most important tools to represent signals in the frequency domain. Fourier transform has found its usage in numerous applications and has led to the development of other transforms. Fourier transform is used in many signal processing applications, such as multimedia, Orthogonal Frequency Division Multiplexing (OFDM), feature detections, and radar.

Four different cases of Fourier transforms can be categorized based on the signal periodicity and the continuous-time/discrete-time variable. Fourier series exploits the property of a signal by keeping in mind that the signal is periodic by nature. Fourier transform extends the periodic boundary by assuming the period of the signal is infinite and Discrete-Time Fourier Transform (DTFT) performs the same analysis in the discrete-time domain. Discrete Fourier Transform (DFT) delivers a frequency analysis tool for the discrete-time signal with the finite number of samples, which is often the case in digital signal processing.

A.1 Fourier Series and Fourier Transform

Given is any real or complex function f of the real variable t. If f is a real periodic function with T as the period, i.e. $f(t+T) = f(t)$, then $f(t)$ can be expanded in a Fourier series

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega t + \sum_{n=1}^{\infty} b_n \sin n\omega t \tag{A.1}$$

with $\omega = 2\pi/T$ as the *fundamental frequency*. The *harmonic frequencies* are given by $n\omega$, with $n = 0, 1, 2, \ldots$. The coefficients a_0, a_n, and b_n are defined as

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\, dt \tag{A.2}$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos n\omega t\, dt, \qquad n \in \mathbb{N} \tag{A.3}$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin n\omega t\, dt, \qquad n \in \mathbb{N} \tag{A.4}$$

A more compact representation of a Fourier series is expressed in terms of complex exponentials

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega t} \tag{A.5}$$

with

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-jn\omega t}\, dt \tag{A.6}$$

Aperiodic signals can be considered as the limit of finite signals over an interval T, with $T \to \infty$. Thus the summation in Fourier series is replaced by a continuous integral over ω and discrete spectrum coefficients c_n are replaced by a continuous function $F(\omega)$

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\, dt = \left\langle e^{j\omega t}, f(t)\right\rangle \tag{A.7}$$

which is a Fourier transform of $f(t)$, and its inverse Fourier transform is given by

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\, d\omega \tag{A.8}$$

The standard textbooks on Fourier theory [**?**, **?**] discuss in detail the exact conditions of the function $f(t)$ so that its inverse exists.

For convenience, the Fourier transform pair is denoted as

$$f(t) \longleftrightarrow F(\omega) \tag{A.9}$$

Briefly, some properties of the Fourier transform are discussed here. The readers are suggested to read [?, ?] for the details and the proofs.

Linearity Fourier transform, as seen in **Eq**. (A.7), is an inner product. Thus, it follows directly from the linearity principle of the inner product

$$\alpha f(t) + \beta g(t) \longleftrightarrow \alpha F(\omega) + \beta G(\omega) \tag{A.10}$$

with α and β as constants.

Symmetry If $F(\omega)$ is the Fourier transform of $f(t)$, then

$$F(t) \longleftrightarrow 2\pi f(-\omega) \tag{A.11}$$

An example of symmetry is the Fourier transform of the rectangular function in time domain is the *sinc* function and the Fourier transform of the *sinc* function is the rectangular function in the frequency domain.

Shifting A shift in time domain by t_0 leads to a multiplication by a phase factor in the Fourier domain

$$f(t - t_0) \longleftrightarrow e^{-j\omega t_0} F(\omega) \tag{A.12}$$

In opposition, a shift in frequency domain by ω_0 leads to a phase factor in the time domain

$$e^{j\omega_0 t} f(t) \longleftrightarrow F(\omega - \omega_0) \tag{A.13}$$

Scaling Scaling in time domain results in inverse scaling in frequency domain.

$$f(at) \longleftrightarrow \frac{1}{|a|} F(\frac{\omega}{a}) \tag{A.14}$$

Moment The nth-order moment of a function is defined as

$$m_n = \int_{-\infty}^{\infty} t^n f(t)\, dt \tag{A.15}$$

The moment theorem of the Fourier transform states

$$(-j)^n m_n = \frac{\partial^n F(\omega)}{\partial^n}\Big|_{\omega=0} \tag{A.16}$$

Convolution Convolution in time domain leads to a multiplication in frequency domain

$$f(t) * g(t) \longleftrightarrow F(\omega)G(\omega) \tag{A.17}$$

By symmetry, convolution in frequeny domain leads to a product in the time domain

$$f(t)g(t) \longleftrightarrow \frac{1}{2\pi} F(\omega) * G(\omega) \tag{A.18}$$

Parseval's Formula Parseval's formula is known as an energy conservation relation formula. The general form of the Parseval's formula for the Fourier transform is

$$\int_{-\infty}^{\infty} f^*(t)g(t)\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} F^*(\omega)G(\omega)\, d\omega \tag{A.19}$$

By setting $g(t) = f(t)$, the formula is reduced to

$$\int_{-\infty}^{\infty} \left|f(t)\right|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left|F(\omega)\right|^2 d\omega \tag{A.20}$$

It states that the total energy computed in the time domain is equal to the total energy computed in frequency domain. The Parseval's theorem allows the energy of the signal to be considered in either the frequency domain or the time domain, and can be interchanged between domains for computation convenience.

A.2 Discrete-Time Fourier Transform

Sampling provides the link between continous-time domain and the discrete-time domain. By sampling the signal with a constant interval, discrete-time signal can be obtained from the continues-time signal. For a function $f(t)$, its sampled version $f_T(t)$ is given by

$$f_T(t) = f(t)s_T(t) \tag{A.21}$$

where

$$s_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \longleftrightarrow S_T(\omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi k}{T}\right) \tag{A.22}$$

are the impulse train with an interval of T and its Fourier transform. Using convolution property given in **Eq**. (A.18), the Fourier transform of $f_T(t)$ is given by

$$\begin{aligned} F_T(\omega) &= F(\omega) * \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{2\pi}{T}\right) \\ &= \frac{1}{T} \sum_{k=-\infty}^{\infty} F\left(\omega - k\frac{2\pi}{T}\right) \end{aligned} \tag{A.23}$$

which states that $F_T(\omega)$ is periodic with period $2\pi/T$ and is obtained by adding up the copies of $F(\omega)$ at every multiple of $2\pi/T$. **Eq**. (A.23) is an equation that leads immediately to the famous sampling theorem of Shannon, Whittaker, and Kotelnikov.

By having a sampled version of the signal in the time domain, the discussion can be continued to the Fourier transform for the discrete-time signal. Analogous to the Fourier series analysis, if a discrete-time signal is aperiodic, it can be considered to be a periodic signal with period $N = \infty$. In the Discrete-Time Fourier Transform (DTFT), the time domain variable is discretized while the frequency variable is still continuous.

Given a summable sequence $f[n]_{n\in\mathbb{Z}}$, its DTFT is defined as

$$F(e^{j\omega}) = \sum_{n=-\infty}^{\infty} f[n]e^{-j\omega n} \tag{A.24}$$

which is 2π-periodic and its inverse as

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(e^{j\omega})e^{j\omega n}\, d\omega \tag{A.25}$$

Comparing **Eq**. (A.5)–**Eq**. (A.6) with **Eq**. (A.24)–**Eq**. (A.25), it is clear that DTFT and Fourier series are dual to each other. Another relation is set up between DTFT and Fourier transform if the sequence $f[n]$ is a sampled version of continuous-time function $f(t)$ at instants nT, that is

$$f[n] = f(nT) \tag{A.26}$$

For the sake of clarity, the Fourier transform pair of the continuous-time signal is denoted as

$$f(t) \longleftrightarrow F_C(\omega) \tag{A.27}$$

and the Fourier transform pair of the sampled version is denoted as

$$f(nT) \longleftrightarrow F_T(\omega) \tag{A.28}$$

$F_T(\omega)$ can be simply related with the Fourier transform of the original function

$$\begin{aligned} F_T(\omega) &= \sum_{n=-\infty}^{\infty} f(nT) e^{-j\omega nT} \\ &= \frac{1}{T} \sum_{k=-\infty}^{\infty} F_C\left(\omega - k\frac{2\pi}{T}\right) \end{aligned} \tag{A.29}$$

Using the definition of DTFT in **Eq**. (A.24) with the consideration at ωT and substituting the discrete-time sequence defined in **Eq**. (A.26), and finally comparing it with **Eq**. (A.29), the DTFT of $f[n]$ leads to

$$\begin{aligned} F(e^{j\omega T}) &= \sum_{n=-\infty}^{\infty} f(nT) e^{-j\omega Tn} \\ &= \frac{1}{T} \sum_{k=-\infty}^{\infty} F_C\left(\omega - k\frac{2\pi}{T}\right) \\ &= F_T(\omega) \end{aligned} \tag{A.30}$$

A.3 Discrete Fourier Transform

For a discrete-time sequence with a period N, that is $f[n] = f[n + \ell N]$, $\ell \in \mathbb{Z}$, the Discrete Fourier Transform (DFT) is defined as

$$F[k] = \sum_{n=0}^{N-1} f[n]\, W_N^{nk} \tag{A.31}$$

where the complex numbers W_N^{nk} are known as the twiddle factor and are defined as

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \tag{A.32}$$

and its inverse (IDFT) is given by

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k]\, W_N^{-nk} \tag{A.33}$$

The relation between DFT and DTFT can be easily obtained by limiting the period of the discrete-time sequence $f[n]$, that is

$$f_0[n] = f[n], \qquad n = 0, 1.., N-1 \tag{A.34}$$

which implies that the DTFT of $f_0[n]$ will be simplified to

$$\begin{aligned} F_0(e^{j\omega}) &= \sum_{n=-\infty}^{\infty} f_0[n]e^{-j\omega n} \\ &= \sum_{n=0}^{N-1} f[n]e^{-j\omega n} \end{aligned} \tag{A.35}$$

It leads to

$$F[k] = F_0(e^{j\omega})\Big|_{\omega=2\pi k/N} \tag{A.36}$$

Because of this repetition, by extending the inverse transform formula for DFT, the original sequence can be recovered by means of generating copies of $f_0[n]$ at integer multiples of N.

$$\begin{aligned} f[n] &= \sum_{\ell=-\infty}^{\infty} f_0[n - \ell N] \\ &= \frac{1}{N}\sum_{k=0}^{N-1} F[k]W_N^{-nk} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} F_0\left[e^{j\frac{2\pi nk}{N}}\right]e^{j\frac{2\pi nk}{N}} \end{aligned} \tag{A.37}$$

The DFT plays an important role in digital signal processing. Nonetheless, the DFT is impractical due to the computation complexity. N complex multiplications and $N-1$ complex additions are needed to compute the transformation on each sample, which leads to N^2 complex multiplications and $N^2 - N$ complex additions for the whole sample range. Fast Fourier Transform (FFT), a much faster algorithm to compute the DFT, was developed by exploiting the symmmetry and the periodicity properties of the complex twiddle factors [**?**, **?**, **?**, **?**].

Appendix B

Orthogonal, Biorthogonal, and Semiorthogonal

In **Sec**. 2.7, the requirement for multiresolution signal analysis was that the wavelet subspace $\mathbf{W_j}$ is complementary of the scaling subspace $\mathbf{V_j}$ in $\mathbf{V_{j+1}}$. In case an *orthogonal decomposition* is considered, following properties must be satisfied:

- The scaling functions $\phi(t-n)$ are orthonormal to each other

$$\int_{-\infty}^{\infty} \phi(t-n)\phi(t-m)\,dt = \delta(m-n) \tag{B.1}$$

- The scaling functions are orthogonal to the wavelets

$$\int_{-\infty}^{\infty} \phi(t-m)\psi(t-n)\,dt = 0 \tag{B.2}$$

- The wavelets $\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k)$ at all scales are orthonormal

$$\int_{-\infty}^{\infty} \psi_{j,k}(t)\psi_{m,n}(t)\,dt = \delta(j-m)\delta(k-n) \tag{B.3}$$

For more details and the proofs of the orthogonalities, see [?].

Observe that the orthonormal wavelets implies that the wavelets are orthonormal with respect to scale as well as with respect to translation in a given scale. On the other hand, the orthonormal scaling function implies that the scaling functions are orthonormal only with respect to translation in a given scale, and not with respect to the scale. It is because of the nested nature of the multiresolution signal analysis.

In the multiresolition signal analysis, **Eq**. (2.77) and **Eq**. (2.78) state that any function $f_j(t) \in \mathbf{V_j}$ and $g_j(t) \in \mathbf{W_j}$ can be represented by the scaling functions $\phi(t)$ and the wavelets $\psi(t)$. That is why these functions are called *synthesis scaling function* and *synthesis wavelet*. Correspondingly, the *analysis scaling function* $\tilde{\phi}(t)$ and the *analysis wavelet* $\tilde{\psi}(t)$ are used to

decompose the signal into their subspaces. If the orthogonalities are satisfied, the synthesis and analysis functions are the same.

The challenge in the orthogonal wavelets is that it is extremely difficult to find bases for analysis and synthesis scaling functions and wavelets that have compactly supported symmetric property. Furthermore, orthonormal scaling functions and wavelets that satisfy the orthogonal conditions have poor time-scale localization. By having different bases for analysis and synthesis functions, these issues can be lifted up. Nonetheless, because perfect reconstruction is also the key in multiresolution signal analysis, the synthesis and analysis pairs, $\{\phi, \psi\}$ and $\{\tilde{\phi}, \tilde{\psi}\}$, must be *dual* to make this work [**?**]. The multiresolution signal decompositions that utilize these types of bases are called *biorthogonal decomposition*.

Basically, the orthogonality requirements are relaxed in order to have a biorthogonal decomposition. However, it is necessary to maintain the requirement that the set of functions $\psi_{j,k}$ or $\tilde{\psi}_{m,n}$ are linearly independent and actually form a basis. In the biorthogonal decomposition, the scaling function $\phi(t)$ and the wavelet $\psi(t)$ must satisfy the duality condition, which implies that

$$\left\langle \psi_{j,k}, \tilde{\psi}_{m,n} \right\rangle = \delta(j-m)\delta(k-n) \tag{B.4}$$

Additionally, if the family is complete in a given space L^2, any function of the space $f(t)$ can be written as

$$f(t) = \sum_j \sum_k \left\langle \psi_{j,k}, f \right\rangle \tilde{\psi}_{j,k}(t) \tag{B.5}$$

$$= \sum_j \sum_k \left\langle \tilde{\psi}_{j,k}, f \right\rangle \psi_{j,k}(t) \tag{B.6}$$

Here, wavelet $\psi_{j,k} \in \mathbf{W_j}$ and its dual $\tilde{\psi}_{j,k} \in \tilde{\mathbf{W}}_\mathbf{j}$. Similarily, a dual scaling function $\tilde{\phi}_{j,k}$ is required to generate another multiresolution signal analysis $\{\tilde{\mathbf{V}}_\mathbf{j}\}$ of L^2. Thus, for $\{\mathbf{V_j}\}$ we have

$$\mathbf{V_{j+1}} = \mathbf{V_j} + \mathbf{W_j} \tag{B.7}$$

$$\mathbf{V}_\ell \cap \mathbf{V_m} = \mathbf{V}_\ell \qquad \ell \le m \tag{B.8}$$

$$\mathbf{W}_\ell \cap \mathbf{W_m} = \{0\} \qquad \ell \ne m \tag{B.9}$$

$$\mathbf{V}_\ell \cap \mathbf{W_m} = \{0\} \qquad \ell \le m \tag{B.10}$$

for $\{\tilde{\mathbf{V}}_\mathbf{j}\}$,

$$\tilde{\mathbf{V}}_\mathbf{j+1} = \tilde{\mathbf{V}}_\mathbf{j} + \tilde{\mathbf{W}}_\mathbf{j} \tag{B.11}$$

$$\tilde{\mathbf{V}}_\ell \cap \tilde{\mathbf{V}}_\mathbf{m} = \tilde{\mathbf{V}}_\ell \qquad \ell \le m \tag{B.12}$$

$$\tilde{\mathbf{W}}_\ell \cap \tilde{\mathbf{W}}_\mathbf{m} = \{0\} \qquad \ell \ne m \tag{B.13}$$

$$\tilde{\mathbf{V}}_\ell \cap \tilde{\mathbf{W}}_\mathbf{m} = \{0\} \qquad \ell \le m \tag{B.14}$$

$$\tag{B.15}$$

and biorthogonalities,

$$\mathbf{W_j} \perp \tilde{\mathbf{V}}_\mathbf{j} \Rightarrow \tilde{\mathbf{V}}_\ell \cap \mathbf{W_m} = \{0\} \qquad \text{for } \ell \leq m \tag{B.16}$$

$$\tilde{\mathbf{W}}_\mathbf{j} \perp \mathbf{V_j} \Rightarrow \mathbf{V}_\ell \cap \tilde{\mathbf{W}}_\mathbf{m} = \{0\} \qquad \text{for } \ell \leq m \tag{B.17}$$

Given a function $f(t) \in L^2$, the decomposition into various resolution scales begins by mapping the function into a sufficiently high resolution subspace $\mathbf{V_J}$. In other word,

$$f(t) \in L^2 \longmapsto f_J(t) = \sum_k v_{J,k}\phi(2^J t - k) \in \mathbf{V_J} \tag{B.18}$$

Recall that

$$\begin{aligned} \mathbf{V_J} &= \mathbf{W_{J-1}} + \mathbf{V_{J-1}} \\ &= \mathbf{W_{J-1}} + \mathbf{W_{J-2}} + \mathbf{V_{J-2}} \\ &= \sum_{n=1}^{N} \mathbf{W_{J-n}} + \mathbf{V_{J-N}} \end{aligned} \tag{B.19}$$

we can write

$$f_J(t) = \sum_{n=1}^{N} g_{J-n}(t) + f_{J-N}(t) \tag{B.20}$$

where f_{J-N} is the coarsest approximation of $f_J(t)$ and

$$f_j(t) = \sum_k v_{j,k}\phi\left(2^j t - k\right) \in \mathbf{V_j} \tag{B.21}$$

$$g_j(t) = \sum_k w_{j,k}\psi\left(2^j t - k\right) \in \mathbf{W_j} \tag{B.22}$$

By using the biorthogonal condition from **Eq**. (B.4), the wavelet coefficients $\{w_{j,k}\}$ can be obtained from **Eq**. (B.22) and written as

$$w_{j,k} = 2^j \int_{-\infty}^{\infty} g_j(t)\tilde{\psi}\left(2^j t - k\right)\, dt \tag{B.23}$$

Because $\tilde{\psi}\left(2^j t - k\right) \in \tilde{\mathbf{W}}_\mathbf{j}$ and $\mathbf{V}_\ell \perp \tilde{\mathbf{W}}_\mathbf{m}$ for $\ell \leq m$, by taking the inner product of **Eq**. (B.20) with $\tilde{\psi}_{j,k}(t)$ and by using the condition from **Eq**. (B.4), we have

$$\begin{aligned} w_{j,k} &= 2^j \int_{-\infty}^{\infty} f_J(t)\tilde{\psi}\left(2^j t - k\right)\, dt \\ &= 2^{j/2} CWT_{\tilde{\psi}} f_J\left(\frac{1}{2^j}, \frac{k}{2^j}\right) \end{aligned} \tag{B.24}$$

Now, it is clear how the dual wavelet $\tilde{\psi}$ gets its name because it is used to analyze the function f_J.

In addition to orthonormal and biorthogonal decomposition, there exist another class of decomposition, called *semiorthogonal decomposition*. In the semiorthogonal decomposition, $\mathbf{V_j} \perp \mathbf{W_j}$. Nevertheless, because the scaling function and the wavelets are nonorthogonal in this system, their duals $\tilde{\phi}$ and $\tilde{\psi}$ are still needed. However, unlike the biorthogonal decomposition, there is no dual space in the semiorthogonal decomposition. This means that $\phi, \tilde{\phi} \in \mathbf{V_j}$ and $\psi, \tilde{\psi} \in \mathbf{W_j}$. In this system, the roles of ϕ, ψ are interchangable with the roles of $\tilde{\phi}$, $\tilde{\psi}$. Semiorthogonal wavelets solve the problem regarding the symmetric property that cannot be satisfied in the orthogonal wavelets. The drawback of the semiorthogonal wavelets is that their duals do not have compact support, which is undesirable since truncating of the coefficients is necessary for real-time signal processing.

Following properties must be satisfied in the semiorthonormal scaling functions and wavelets:

- The scaling function $\phi(t)$ and its dual $\tilde{\phi}(t)$ are orthonormal to each other

$$\left\langle \phi(t-n), \tilde{\phi}(t-m) \right\rangle = \delta(m-n) \tag{B.25}$$

- The wavelet $\psi(t)$ and its dual $\tilde{\psi}(t)$ are orthogonal to each other

$$\left\langle \psi(t-n), \tilde{\psi}(t-m) \right\rangle = 0 \tag{B.26}$$

The relation between $\{\phi, \psi\}$ and $\{\tilde{\phi}, \tilde{\psi}\}$ are defined as

$$\tilde{\Phi}(\omega) = \frac{\Phi(\omega)}{E_{\phi}(e^{j\omega})} \tag{B.27}$$

and

$$\tilde{\Psi}(\omega) = \frac{\Psi(\omega)}{E_{\psi}(e^{j\omega})} \tag{B.28}$$

with

$$\begin{aligned} E_f\left(e^{j\omega}\right) &= \sum_{k=-\infty}^{\infty} |F(\omega + 2\pi k)|^2 \\ &= \sum_{k=-\infty}^{\infty} A_f(k) e^{j\omega k} \end{aligned} \tag{B.29}$$

where $A_f(t)$ is the autocorrelation function of $f(t)$. For details and the proof of the orthonormalities, see [?].

Appendix C

Factoring Wavelet Filters to Lifting Steps

The goal of the lifting steps is to divide the computation cost by using several predictions and updates. This appendix merely restates the contribution from Daubechies and Sweldens in [?, ?], who have detailed the algorithm for factoring the polyphase matrix into lifting steps.

Given a 2×2 matrix

$$\mathbf{P}(z) = \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}_e(z) & \tilde{G}_o(z) \end{bmatrix} \tag{C.1}$$

where $\tilde{H}_e(z)$, $\tilde{H}_o(z)$, $\tilde{G}_e(z)$, $\tilde{G}_o(z)$ are Laurent polynomials and

$$\Delta_{\mathbf{P}}(z) = 1 \tag{C.2}$$

there exist a constant $K \neq 0$ and Laurent polynomials $p_i(z)$ and $u_i(z)$ such that

$$\mathbf{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{M} \begin{bmatrix} 1 & u_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ p_i(z) & 1 \end{bmatrix} \tag{C.3}$$

Because the transform is invertible, $\mathbf{P}^{-1}$ exists and $\Delta_{\mathbf{P}}(z) = 1/(cz^n)$ for some $a \neq 0$ and n. Consequently,

$$\Delta \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ cz^n\tilde{G}_e(z) & cz^n\tilde{G}_o(z) \end{bmatrix} = 1 \tag{C.4}$$

The factorization is performed by using the Euclidean algorithm as an algorithm for finding the greatest common divisor of two polynomials. This algorithm is based on an iteration approach in order to find zero remainder as the solution. Given two Laurent polynomials $a(z)$ and $b(z)$ such that $|a(z)| \geq |b(z)|$. There exist a Laurent polynomial $q(z)$ as the quotient with $\left|q(z)\right| = |a(z)| - |b(z)|$ and a Laurent polynomial $r(z)$ as the remainder with $|r(z)| < |b(z)|$ such that

$$a(z) = b(z)q(z) + r(z) \tag{C.5}$$

With the help of these notations

$$q(z) = a(z) \,/\, b(z) \tag{C.6}$$

$$r(z) = a(z) \,\%\, b(z) \tag{C.7}$$

the process can be iterated to have $r(z) = 0$ at the end. By setting initial polynomials $a_0(z) = a(z)$ and $b_0(z) = b(z)$, the following iteration starts from $n = 0$

$$a_{n+1}(z) = b_n(z) \tag{C.8}$$

$$b_{n+1}(z) = a_n(z) \,\%\, b_n(z) \tag{C.9}$$

After N steps, we have

$$a_N(z) = b_{n-1}(z) \tag{C.10}$$

$$0 = b_n(z) = a_{N-1}(z) - b_{N-1}(z)q_N(z) \tag{C.11}$$

In the matrix form,

$$\begin{bmatrix} a_N(z) \\ 0 \end{bmatrix} = \prod_{n=N}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -q_n(z) \end{bmatrix} \begin{bmatrix} a(z) \\ b(z) \end{bmatrix} \tag{C.12}$$

Because

$$\begin{bmatrix} 0 & 1 \\ 1 & -q_n(z) \end{bmatrix}^{-1} = \begin{bmatrix} q_n(z) & 1 \\ 1 & 0 \end{bmatrix} \tag{C.13}$$

we have

$$\begin{bmatrix} a(z) \\ b(z) \end{bmatrix} = \prod_{n=1}^{N} \begin{bmatrix} q_n(z) & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_N(z) \\ 0 \end{bmatrix} \tag{C.14}$$

Applying **Eq**. (C.14) to the low-pass filter $\tilde{H}(z)$ leads to

$$\begin{bmatrix} \tilde{H}_e(z) \\ \tilde{H}_o(z) \end{bmatrix} = \prod_{n=1}^{N} \begin{bmatrix} q_n(z) & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} Kz^c \\ 0 \end{bmatrix} \tag{C.15}$$

By multiplying both sides with z^{-c}, we have

$$\begin{bmatrix} \tilde{H}'_e(z) \\ \tilde{H}'_o(z) \end{bmatrix} = \begin{bmatrix} z^{-c}\tilde{H}_e(z) \\ z^{-c}\tilde{H}_o(z) \end{bmatrix} = \prod_{n=1}^{N} \begin{bmatrix} q_n(z) & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix} \tag{C.16}$$

This means that it is always possible to end up with $a_N(z) = K$.

With a slight arrangement

$$\begin{bmatrix} q(z) & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & q(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ q(z) & 0 \end{bmatrix} \tag{C.17}$$

and separating odds and evens, it leads to

$$\begin{bmatrix} \tilde{H}_e(z) \\ \tilde{H}_o(z) \end{bmatrix} = \prod_{n=1}^{N/2} \begin{bmatrix} 1 & q_{2n-1}(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ q_{2n}(z) & 1 \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix} \tag{C.18}$$

Replacing

$$\begin{bmatrix} K \\ 0 \end{bmatrix} \text{ with } \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix}$$

leads to

$$\begin{aligned} \mathbf{P}'(z) &= \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}'_e(z) & \tilde{G}'_o(z) \end{bmatrix} \\ &= \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{n=M}^{1} \begin{bmatrix} 1 & q_{2n}(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ q_{2n-1}(z) & 1 \end{bmatrix} \end{aligned} \tag{C.19}$$

Because $\Delta_{\mathbf{P}}(z) = 1$, $\tilde{G}'(z)$ is related with $\tilde{H}(z)$ by

$$\tilde{G}'(z) = \tilde{G}(z) + \tilde{H}(z)t(z) \tag{C.20}$$

where $t(z)$ is a Laurent polynomial to make $\Delta_{\mathbf{P}'}(z) = 1$. Thus we have

$$\begin{aligned} \mathbf{P}'(z) &= \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}'_e(z) & \tilde{G}'_o(z) \end{bmatrix} \\ &= \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}_e(z) + \tilde{H}_e(z)t(z) & \tilde{G}_o(z) + \tilde{H}_o(z)t(z) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ t(z) & 1 \end{bmatrix} \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}_e(z) & \tilde{G}_o(z) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ t(z) & 1 \end{bmatrix} \mathbf{P}(z) \end{aligned} \tag{C.21}$$

The original high-pass filter $\tilde{G}(z)$ can be resolved by applying **Eq**. (C.21) to **Eq**. (C.19). Rewriting and rearranging **Eq**. (C.20) for $\tilde{G}$ in matrix form,

$$\begin{bmatrix} \tilde{G}_e(z) \\ \tilde{G}_o(z) \end{bmatrix} = \begin{bmatrix} \tilde{H}_e(z) & \tilde{G}'_e(z) \\ \tilde{H}_o(z) & \tilde{G}'_o(z) \end{bmatrix} \begin{bmatrix} -t(z) \\ 1 \end{bmatrix} \tag{C.22}$$

the solution for $t(z)$ is given by using the inverse of the 2×2 matrix,

$$\begin{bmatrix} -t(z) \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{G}'_o(z) & -\tilde{G}'_e(z) \\ -\tilde{H}_o(z) & \tilde{H}_e(z) \end{bmatrix} \begin{bmatrix} \tilde{G}_e(z) \\ \tilde{G}_o(z) \end{bmatrix} \tag{C.23}$$

which leads to

$$t(z) = \tilde{G}'_e(z)\tilde{G}_o(z) - \tilde{G}'_o(z)\tilde{G}_e(z) \tag{C.24}$$

Rearranging **Eq**. (C.21) for $\mathbf{P}(z)$, we have

$$\mathbf{P}(z) = \begin{bmatrix} 1 & 0 \\ -t(z) & 1 \end{bmatrix} \mathbf{P}'(z) \tag{C.25}$$

With

$$\begin{bmatrix} 1 & 0 \\ -t(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -K^2 t(z) & 1 \end{bmatrix} \tag{C.26}$$

The solution to **Eq**. (C.25) is given by replacing $\mathbf{P}'(z)$ with **Eq**. (C.19) and using the help from **Eq**. (C.26).

$$\mathbf{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -K^2 t(z) & 1 \end{bmatrix} \prod_{n=M}^{1} \begin{bmatrix} 1 & q_{2n}(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ q_{2n-1}(z) & 1 \end{bmatrix} \tag{C.27}$$

By suitable reindexing of the $q_n(z)$ polynomials and at the same time making $K^2 t(z)$ one of them, the Laurent polynomials $a_i(z)$ and $b_i(z)$ from **Eq**. (C.3) can be determined.

References

[1] J. AGBINYA. Discrete wavelet transform techniques in speech processing. In *Proc. of the IEEE TENCON. Digital Signal Processing Applications, TENCON '96*, volume 2, pages 514–519, 1996.

[2] J. ALLEN. Short term Spectral Analysis, Synthesis, and Modification by Discrete Fourier Transform. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 25:235–238, 1977.

[3] K. ANDRA, C. CHAKRABARTI, and T. ACHARYA. A VLSI architecture for lifting-based forward and inverse wavelet transform. *IEEE Trans. Signal Process.*, 50(4):966–977, 2002.

[4] F. ARGUELLO, J. LOPEZ, M. TRENAS, and E. ZAPATA. Architecture for Wavelet Packet Transform Based on Lifting Steps. *J. Parallel Computing*, 28:1023–1037, 2002.

[5] S. AROUTCHELVAME and K. RAAHEMIFAR. Architecture of Wavelet Packet Transform for 1-D Signal. In *Proc. of the IEEE Canadian Conf. on Elect. and Comput. Eng.*, 2005.

[6] G. ATY, A. HUSSEIN, I. ASHOUR, and M. MONES. High-speed, area-efficient FPGA-based floating-point multiplier. In *Proc. of the 15th Intl. Conference on Microelectronics*, 2003.

[7] S. BARUA, J. CARLETTA, K. KOTTERI, and A. BELL. An Efficient Architecture for Lifting-based Two-Dimensional Discrete Wavelet Transforms. In *Proc. of the Great Lakes Symposium on VLSI, GLSVLSI '04*, 2004.

[8] S. BARUA, K. KOTTERI, A. BELL, and J. CARLETTA. Optimal quantized lifting coefficients for the 9/7 wavelet [image compression applications]. In *Proc. of the IEEE Intl. Conference on Acoustics, Speech, and Signal Processing, ICASSP '04*, volume 5, 17–21 May 2004.

[9] A. BEAUMONT-SMITH, N. BURGESS, S. LEFRERE, and C. LIM. Reduced latency IEEE floating-point standard adder architectures. In *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, 14-16 April 1999.

[10] M. BELLANGER and J. DAGUET. TDM-FDM Transmultiplexer: Digital Polyphase and FFT. *IEEE Trans. Communication*, 22:1199–1204, 1974.

[11] K. BLINOWSKA and P. DURKA. Introduction to wavelet analysis. *Br J Audiol*, 31(6):449–459, Dec 1997.

[12] R. N. BRACEWELL. *The Fourier Transform and its Applications*. McGraw-Hill, New York, 1986.

[13] J. BRUGUERA and T. LANG. Leading-one prediction scheme for latency improvement in single datapath floating-point adders. In *Proc. of the Intl. Conference on Computer Design: VLSI in Computers and Processors*, 5-7 Oct. 1998.

[14] J. BRUGUERA and T. LANG. Leading-one prediction with concurrent position correction. *IEEE Transactions on Computers*, 48:1083–1097, 1999.

[15] A. BULTHEEL. Wavelets with applications in signal and image processing, 2003.

[16] C. S. BURRUS, R.A.GOPINATH, and H. GUO. *Introduction to Wavelets and Wavelet Transforms: A Primer*. Prentice Hall, 1997.

[17] R. CALDERBANK, I. DAUBECHIES, W. SWELDENS, and B.-L. YEO. Lossless image compression using integer to integer wavelet transforms. In *Proc. of the Intl. Conference on Image Processing, ICIP '97*, volume 1, pages 596–599. IEEE Press, 1997.

[18] R. CALDERBANK, I. DAUBECHIES, W. SWELDENS, and B.-L. YEO. Wavelet transforms that map integers to integers. *Appl. Comput. Harmon. Anal.*, 5(3):332–369, 1998.

[19] B. CARNERO and A. DRYGAJLO. Perceptual speech coding and enhancement using frame-synchronized fast wavelet packet transform algorithms. *IEEE Trans. Signal Process.*, 47:1622–1634, 1999.

[20] C. CHAKRABARTI, M. VISHWANATH, and R. OWENS. Architectures for Wavelet Transforms. In *Proc. of the IEEE VLSI Signal Processing Workshop*, 1993.

[21] D. C. CHAMPENEY. *A Handbook of Fourier Theorems*. Cambridge University Press, Cambridge, UK, 1987.

[22] Y. T. CHAN. *Wavelet Basics*. Kluwer Academic Publishers, 1994.

[23] Y. CHENG, Y. LI, and R. ZHAO. A Parallel Image Fusion Algorithm Based on Wavelet Packet. In *Proc. of the Intl. Conf. on Signal Processing*, 2006.

[24] C. CHRISTOPOULOS, A. SKODRAS, and T. EBRAHIMI. The JPEG2000 still image coding system: an overview. *IEEE Trans. Consum. Electron.*, 46(4):1103–1127, 2000.

[25] C. CHRYSAFIS and A. ORTEGA. Line-Based, Reduced Memory, Wavelet Image Compression. *IEEE Trans. on Image Processing*, 9:378–389, 2000.

[26] A. COHEN, I. DAUBECHIES, and J.-C. FEAUVEAU. Biorthogonal bases of compactly supported wavelets. *Communication on Pure and Applied Math*, 45:485–56, 1992.

[27] J. W. COOLEY and J. W. TUKEY. An Algorithm for Machine Calculation of Complex Fourier Series. *Math. Computation*, pages 297–301, 1965.

[28] A. CROISIER, D. ESTEBAN, and C. GALAND. Perfect Channel Splitting by the use of Interpolation/Decimation/Tree Decomposition Techniques. In *Intl. Conf. on Information, Sciences, and Systems*, pages 443–446, 1976.

[29] P. DANG and P. CHAU. Reduce complexity hardware implementation of discrete wavelet transform for JPEG 2000 standard. In *Proc. of the IEEE Intl. Conference on Multimedia and Expo ICME '02*, volume 1, pages 321–324, 26–29 Aug. 2002.

[30] I. DAUBECHIES. Orthonormal Bases of Compactly Supported Wavelets. *Communication on Pure and Applied Math*, 41:909–996, 1988.

[31] I. DAUBECHIES. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. on Information Theory*, 36:961–1005, 1990.

[32] I. DAUBECHIES. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics No. 61. Society for Industrial & Applied Mathematics, 1992.

[33] I. DAUBECHIES and W. SWELDENS. Factoring Wavelet Transforms into Lifting Steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[34] G. DILLEN, B. GEORIS, J. LEGAT, and O. CANTINEAU. Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000. *IEEE Trans. Circuits Syst. Video Technol.*, 13(9):944–950, Sept. 2003.

[35] G. EVEN, S. MUELLER, and P.-M. SEIDEL. A dual mode IEEE multiplier. In *Proc. of the Second Annual IEEE Intl. Conference on Innovative Systems in Silicon*, 1997.

[36] G. EVEN and W. PAUL. On the design of IEEE compliant floating point units. *IEEE Trans. Comput.*, 49(5):398–413, May 2000.

[37] M. FARAHANI and M. ESHGHI. Architecture of a Wavelet Packet Transform Using Parallel Filters. In *Proc. of the IEEE Conf. on Applied Electronics*, 2006.

[38] K. FERENS and W. KINSNER. Adaptive wavelet subband coding for music compression. In W. KINSNER, ed., *Proc. of the Data Compression Conference, DCC '95*, 1995.

[39] N. FLIEGE. *Multirate Digital Signal Processing*. John Wiley & Sons, Inc., 1994.

[40] D. GABOR. Theory of Communication. *IEE Journal*, pages 429–457, 1946.

[41] R. GANDHIRAJ and P. SATHIDEVI. Auditory-Based Wavelet Packet Filterbank for Speech Recognition Using Neural Network. In *Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on*, pages 666–673, 2007.

[42] G. GERWIG and M. KROENER. Floating-Point Unit in Standard Cell Design with 116 Bit Wide Dataflow. In *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, 1999.

[43] J. GOSWAMI and A. CHAN. *Fundamentals of Wavelets: Theory, Algorithms, and Applications*. John Wiley & Sons, Incorporated, 1999.

[44] M. GREEN and A. ZOUBIR. Selection of a Time-Varying Quadratic Volterra Model Using a Wavelet Packet Basis Expansion. *IEEE Trans. Signal Process.*, 52:2721–2728, 2004.

[45] S. GRGIC, K. KERS, and M. GRGIC. Image compression using wavelets. In K. KERS, ed., *Proc. of the IEEE Intl. Symposium on Industrial Electronics, ISIE '99*, volume 1, 1999.

[46] A. GRZESZCZAK, M. MANDAL, S. PANCHANATHAN, and T. YEAP. VLSI Implementation of Discrete Wavelet Transform. *IEEE Trans. on VLSI Systems*, 4:412–433, 1996.

[47] Y. HAGIHARA, S. INUI, F. OKAMOTO, M. NISHIDA, T. NAKAMURA, and H. YAMADA. Floating-point datapaths with online built-in self speed test. *IEEE J. Solid-State Circuits*, 32(3):444–449, March 1997.

[48] S. HE and M. TORKELSON. Design and Implementation of a 1024-point Pipeline FFT Processor. In *IEEE Custom Integrated Circuits Conference*, pages 131–134, 1998.

[49] C. HEIKES and G. COLON-BONET. A dual floating point coprocessor with an FMAC architecture. In *Proc. of the IEEE Intl. Solid-State Circuits Conference Digest of Technical Papers. 43rd ISSCC*, pages 354–355,472, 8–10 Feb. 1996.

[50] E. HOKENEK and R. K. MONTOYE. Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit. *IBM Journal of Research and Development*, 34:71–77, 1990.

[51] IEEE STANDARD BOARD AND ANSI. IEEE standard for binary floating-point arithmetic, 1985. IEEE Std 754-1985.

[52] N. JAYANT and P. NOLL. *Digital Coding for Waveforms*. Prentice Hall, Englewood-Cliffs, 1984.

[53] Y. KAISHENG and C. ZHIGANG. A wavelet filter optimization algorithm for speech recognition. In *Intl. Conference on Communication Technology Proc., ICCT '98*, volume 2, page 5, 22-24 Oct. 1998.

[54] Y. KANG. Low-power Design of Wavelet Processors. In *Proc. of the SPIE*, pages 1800–1806, 1993.

[55] G. KARLSSON and M. VETTERLI. Theory of Two-Dimensional Multirate Filter Banks. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 38:925–937, 1990.

[56] R. KERSHAW, L. BAYS, R. FREYMAN, J. KLINIKOWSKI, C. MILLER, K. MONDAL, H. MOSCOVITZ, W. STOCKER, L. TRAN, W. HAYS, J. BODDIE, E. FIELDS, C. GAREN, and J. TOW. A programmable digital signal processor with 32b floating point arithmetic. In *Proc. of the IEEE Intl. Solid-State Circuits Conference. Digest of Technical Papers*, 1985.

[57] K. KOTTERI, S. BARUA, A. BELL, and J. CARLETTA. A comparison of hardware implementations of the biorthogonal 9/7 DWT: convolution versus lifting. *IEEE Trans. Circuits Syst. II*, 52(5):256–260, May 2005.

[58] J. KOVAČEVIĆ and M. VETTERLI. Nonseparable Multidimensional Perfect Reconstruction Filter Banks and Wavelet Bases for $\mathbb{R}^n$. *IEEE Trans. of Information Theory, Special Issue on Wavelet Transforms and Multiresolutional Signal Analysis*, 38:533–555, 1992.

[59] J. KOWALESKI, G. WOLRICH, T. FISCHER, R. DUPCAK, P. KROESEN, T. PHAM, and A. OLESIN. A dual execution pipelined floating-point CMOS processor. In *Proc. of the IEEE Intl. Solid-State Circuits Conference Digest of Technical Papers. 43rd ISSCC*, pages 358–359,473, 8–10 Feb. 1996.

[60] W. KRANDICK and J. JOHNSON. Efficient multiprecision floating point multiplication with optimal directional rounding. In *Proc. of the Symposium on Computer Arithmetic*, pages 228–233, 29 June–2 July 1993.

[61] F. KURTH and M. CLAUSEN. Filter Bank Tree and M-band Wavelet Packet Algorithms in Audio Signal Processing. *IEEE Trans. on Signal Processing*, 47:549–554, 1999.

[62] T. LE. *Algorithmic and Architectural Transformations for Flexible Signal Processing*. PhD thesis, TU Darmstadt, 2000.

[63] C.-J. LIAN, K.-F. CHEN, H.-H. CHEN, and L.-G. CHEN. Lifting Based Discrete Wavelet Transform Architecture for JPEG2000. In K.-F. CHEN, ed., *Proc. of the IEEE Intl. Symposium on Circuits and Systems, ISCAS '01*, volume 2, pages 445–448, 2001.

[64] H. LIAO, M. MANDAL, and B. COCKBURN. Novel architectures for the lifting-based discrete wavelet transform. In M. MANDAL, ed., *Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering, CCECE '02*, volume 2, pages 1020–1025, 2002.

[65] C.-C. LIU, Y.-H. SHIAU, and J.-M. JOU. Design and Implementation of a Progressive Image Coding Chip based on the Lifted Wavelet Transform. In *Proc. of the IEEE VLSI Design/CAD Symposium*, 2000.

[66] S. MALLAT. *Multiresolution Representation and Wavelets*. PhD thesis, University of Pennsylvania, Philadelphia, 1988.

[67] S. MALLAT. A Theory for Multiresolution Approximations and Wavelet Orthonormal Bases of $\ell_2(r)$. *IEEE Trans. Pattern Recognition and Machine Intelligent*, 11:674–693, 1989.

[68] S. MALLAT. A Theory of Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Trans. Pattern Anal. Machine Intell.*, 11:674–693, 1989.

[69] S. MALLAT, ed. *A Wavelet Tour of Signal Processing*. Academic Press, Incorporated, 1998.

[70] F. MARINO, D. GUEVORKIAN, and J. ASTOLA. Highly Efficient High-Speed/Low-Power Architectures for the 1-D Discrete Wavelet Transform. *IEEE Trans. on Circuits and Systems*, 47:1492–1502, 2000.

[71] M. MARTINA, G. MASERA, G. PICCININI, and M. ZAMBONI. A VLSI architecture for IWT (Integer wavelet Transform). In G. MASERA, ed., *Proc. of the 43rd IEEE Midwest Symposium on Circuits and Systems*, volume 3, 2000.

[72] M. MEHENDALE, S. ROY, S. SERLEKAR, and G. VENKETESH. Coefficient Transformations for Area-Efficient Implementation of Multiplier-less FIR Filters. In *Proc. of IEEE on VLSI Design*, 1998.

[73] A. MERTINS. *Signal Analysis Wavelets: Filter Banks, Time-Frequency Transforms and Applications*. John Wiley & Sons, 1999.

[74] Y. MEYER. *Wavelets and Operators*. Press Syndicate of the University of Cambridge, 1992.

[75] Y. MEYER. *Wavelets Algorithms and Applications*. Society for Industrial & Applied Mathematics, 1993.

[76] F. MINTZER. Filter for Distortion-free Two-band Multirate Filter Banks. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 33:626–630, 1985.

[77] S. H. NAWAB and T. QUARTIERI. *Advanced Topics in Signal Processing*, chapter Short-Time Fourier Transform, pages 289–337. Prentice-Hall, Englewood Cliffs, N.J., 1988.

[78] T. NGUYEN and P. VAIDYANATHAN. Two-Channel Perfect Reconstruction FIR QMF structure which yields Linear Phase Analysis and Synthesis Filters. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 37:676–690, 1989.

[79] T. PAINTER and A. SPANIAS. Perceptual Coding of Digital Audio. In *Proc. IEEE*, pages 451–515, 2000.

[80] S. PAN and R. PARK. New Systolic Arrays for Computation of the 1-D Discrete Wavelet Transform. In *Proc. of the IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, volume 5, pages 4113–4116, 1997.

[81] A. PAPOULIS. *The Fourier Integral and its Applications*. McGraw-Hill, New York, 1962.

[82] K. PARHI. Video Data Format Converters using Minimum Number of Registers. *IEEE Trans. Circuits System Video Tech.*, 38:255–267, 1992.

[83] K. PARHI and T. NISHITANI. VLSI Architecture for Discrete Wavelet Transforms. *IEEE Trans. VLSI System*, pages 191–202, 1993.

[84] G. PAYA, M. PEIRO, F. BALLESTER, V. HERRERO, and F. MORA. Lifting Folded Pipelined Discrete Wavelet Packet Transform Architecture. In *Proc. of SPIE*, 2003.

[85] R. PILLAI, D. AL-KHALILI, and A. AL-KHALILI. Low power architecture for floating point MAC fusion. In *IEE Proc. Computers and Digital Techniques*, volume 147, pages 288–296, 2000.

[86] M. PORTNOFF. Representation of Digital Signals and Systems based on Short-Time Fourier Analysis. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 28:55–69, 1980.

[87] A. D. POULARIKAS, ed. *The Transforms and Applications Handbook*. CRC Press LLC, 2000.

[88] L. R. RABINER and B. GOLD. *Theory and Application of Digital Signal Processing*. Prentice-Hall, 1975.

[89] P. RADEMACHER, B. CANTRELL, and G. TAVIK. Clutter Filtering and Processing Techniques for EMI Detection and Angle Measurement in Pulse Doppler Radars. In *Proc. of the IEEE Nat. Radar Conference*, pages 273–278, 1996.

[90] S. RAIESDANA, M. SHAMSOLLAHI, M. HASHEMI, and I. REZAZADEH. Wavelet Packet Decomposition of a New Filter-Based on Underlying Neural Activity for ERP Classification. In *Proc. of IEEE. Engineering in Medicine and Biology Society*, 2007.

[91] P.-M. SEIDEL and G. EVEN. Delay-optimized implementation of IEEE floating-point addition. *IEEE Transactions on Computers*, 53:97–113, 2004.

[92] L. SENHADJI, G. CARRAULT, and J. BELLANGER. Interictal EEG Spike detection: A New Framework based on Wavelet Transform. In *Proc. of the IEEE-SP Intl. Symp. Time-Frequency Time-Scale Analysis*, pages 548–551, 1996.

[93] Y.-H. SEO and D.-W. KIM. A New VLSI Architecture of Lifting-Based DWT. *Lecture Notes in Computer Science*, 3985/2006:146–151, 2006.

[94] Y.-H. SEO and D.-W. KIM. VLSI Architecture of Line-Based Lifting Wavelet Transform for Motion JPEG2000. *IEEE Journal of Solid-State Circuits*, 42:431–440, 2007.

[95] P.-L. SHUI, Z.-F. ZHOU, and J.-X. LI. Image Denoising Algorithm via Best Wavelet Packet Base using Wiener Cost Function. In *Proc. of the IET Image Processing*, 2007.

[96] M. SMITH and T. BARNWELL III. A Procedure for Designing Exact Reconstruction Filter Banks for Tree Structured Sub-band Coders. In *Proc. of the IEEE Intl. Conf. Acoustic, Speech, and SIgnal Processing, San Diego, CA.*, 1984.

[97] M. SMITH and T. BARNWELL III. Exact Reconstruction for Tree-Structured Subband Coders. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 34:431–441, 1986.

[98] M. STEINER and F. KRETSCHMER, JR. Missing Radar Pulse Clutter Processing. In *Proc. of the IEEE Nat. Radar Conference*, pages 112–116, 1991.

[99] M. STOKSIK, R. LANE, and D. NGUYEN. Accurate Synthesis of Fractional Brownian Motion using Wavelets. *Electronic Letter*, 30:383–384, 1996.

[100] R. STORN. Radix-2 FFT-Pipeline Architecture with Reduced Noise-to-Signal Ratio. In *Proc. IEEE Vision, Image, and Signal Processing*, pages 81–86, 1994.

[101] G. STRANG and T. NGUYEN. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[102] H. SUZUKI, H. MORINAKA, H. MAKINO, Y. NAKASE, K. MASHIKO, and T. SUMI. Leading-zero anticipatory logic for high-speed floating point addition. *IEEE J. Solid-State Circuits*, 31(8):1157–1164, Aug. 1996.

[103] W. SWELDENS. The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions. *Wavelet Applications in Signal and Image Processing*, 3:68–79, 1995.

[104] W. SWELDENS. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.

[105] B. USEVITCH. A tutorial on modern lossy wavelet image compression: foundations of JPEG2000. *IEEE Signal Process. Mag.*, 18(5):22–35, 2001.

[106] M. VETTERLI. Multidimensional Subband Coding: Some Theory and Algorithms. *Signal Processing*, 6:97–112, 1984.

[107] M. VETTERLI. Filter Banks allowing Perfect Reconstruction. *Signal Processing*, 10:219–244, 1986.

[108] M. VETTERLI and J. KOVACEVIC. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[109] M. VETTERLI and D. LEGALL. Perfect Reconstruction FIR Filter Banks: Some Properties and Factorizations. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 37, 1989.

[110] P. VIADYANATHAN. Theory and Design of M-Channel Maximally Decimated Quadrature Mirror Filters with arbitrary M, having the Perfect Reconstruction Property. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 35:476–492, 1987.

[111] P. VIADYANATHAN. *Multirate Systems and Filter Banks*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[112] P. VIADYANATHAN and Z. DOĞANATA. The Role of Lossless Systems in Modern Digital Signal Processing: A Tutorial. *IEEE Trans. Education*, 32:181–197, 1989.

[113] E. VISCITO and J. ALLEBACH. The Analysis and Design of Multidimensional FIR Perfect Reconstruction Filter Banks for Arbitrary Sampling Lattices. *IEEE Trans. Circuit and Systems*, 38:29–42, 1991.

[114] M. VISHWANATH. Discrete Wavelet Transform in VLSI. In *Proc. of the IEEE. Intl. Conf. Appl. Specific Array Processors*, 1992.

[115] J. S. WALTER. *Fourier Analysis*. Oxford University Press, New York, 1988.

[116] C. WANG and W. GAN. Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Packet Transform. *IEEE Trans. Circuits Syst. II*, 54(5):422–426, 2007.

[117] B.-F. WU and C.-F. LIN. A High-Performance and Memory-Efficient Pipeline Architecture for the 5/3 and 9/7 Discrete Wavelet Transform of JPEG2000 Codec. *IEEE Trans. on Circuits and Systems for Video Technology*, 15:1615–1628, 2005.

[118] Y. YANG and C. XU. A Wavelet Packet Based Block-Partitioning Image Coding Algorithm with Rate-Distortion Optimization. In *Proc. of the IEEE Intl. Conf. on Image Processing*, 2005.

[119] R. YU and G. ZYNER. 167 Mhz radix-4 floating point multiplier. In *Proc. of the 12th Symposium on Computer Arithmetic*, 1995.

List of Publications

[120] A. GUNTORO and M. GLESNER. A Flexible Floating-Point Wavelet Processor. In *Proc. of the Intl. Conference on Signal-Image Technology and Internet-Based Systems, SITIS '08*, Dec. 2008.

[121] A. GUNTORO and M. GLESNER. A Lifting-Based DWT and IDWT Processor with Multi-Context Configuration and Normalization Factor. In *Proc. of the 18th IEEE Intl. Conference on Field Programmable Logic and Applications, FPL '08*, pages 479–482, Sep. 2008.

[122] A. GUNTORO and M. GLESNER. A Lifting-Based DWT and IDWT Processor with Support for Higher Order Wavelet Filters. In *Proc. of the Intl. Conference on Very Large Scale Integration, VLSI-SoC '08*, pages 73–79, Oct. 2008.

[123] A. GUNTORO and M. GLESNER. Configurable VLSI Architecture of a 3-Input Floating-Point Adder. In *Proc. of the Intl.Conference on Signals and Electronic Systems, ICSES '08*, pages 121–124, Sep. 2008.

[124] A. GUNTORO and M. GLESNER. High-Performance Floating-Point VLSI Architecture of Lifting-Based Forward and Inverse Wavelet Transforms. In *Proc. of the Intl. Asia-Pacific Conference on Circuits and Systems, APCCAS '08*, Dec. 2008.

[125] A. GUNTORO and M. GLESNER. High-Performance FPGA-based Floating-Point Adder with Three Inputs. In *Proc. of the 18th IEEE Intl. Conference on Field Programmable Logic and Applications, FPL '08*, pages 627–630, Sep. 2008.

[126] A. GUNTORO and M. GLESNER. High-Precision VLSI Architecture of Lifting-Based Forward and Inverse Wavelet Transforms. In *Proc. of the Intl. Conference on Design and Architectures for Signal and Image Processing, DASIP '08*, Nov. 2008.

[127] A. GUNTORO and M. GLESNER. Low-Latency VLSI Architecture of a 3-Input Floating-Point Adder. In *Proc. of the Intl. Asia-Pacific Conference on Circuits and Systems, APCCAS '08*, Dec. 2008.

[128] A. GUNTORO and M. GLESNER. Novel Approach on Lifting-Based DWT and IDWT Processor with Multi-Context Configuration to Support Different Wavelet Filters. In *Proc. of the 19th IEEE Intl. Conference Application-specific Systems, Architectures and Processors, ASAP '08*, pages 299–304, July 2008.

[129] A. GUNTORO and M. GLESNER. Resolving Longitudinal Amplitude and Phase Information of Two Continuous Data Streams for High-Speed and Real-Time Processing. In *Advances in Radio Science - Kleinheubacher Berichte*, Sep. 2008.

[130] A. GUNTORO and M. GLESNER. A Flexible Floating-Point Wavelet Transform and Wavelet Packet Processor. In *Proc. of the Intl. Conference on Design, Automation and Test in Europe, DATE '09*, Apr. 2009.

[131] A. GUNTORO and M. GLESNER. *VLSI-SOC: Design Methodologies for SoC and SiP*, chapter An Lifting-Based Discrete Wavelet Transform and Discrete Wavelet Packet Processor with Support for Higher Order Wavelet Filters. Springer Publishers, 2009.

[132] A. GUNTORO, H.-P. KEIL, and M. GLESNER. Configurable VLSI Architecture of a General Purpose Lifting-based Wavelet Processor. In *Proc. of the Intl. Conference on Signal Processing and Multimedia Applications, SIGMAP '08*, pages 69–75, July 2008.

[133] A. GUNTORO, H.-P. KEIL, and M. GLESNER. *ICETE 2008*, chapter High-Speed Configurable VLSI Architecture of a General Purpose Lifting-Based Discrete Wavelet Processor, pages 318–330. Springer-Verlag Berlin Heidelberg, 2009.

[134] A. GUNTORO, M. MOMENI, H.-P. KEIL, and M. GLESNER. High-Performance Floating-Point VLSI Architecture of a Lifting-Based Wavelet Processor. In *Proc. of the Intl. Conference on Signals and Electronic Systems, ICSES '08*, pages 35–38, Sep. 2008.

[135] A. GUNTORO, P. ZIPF, O. SOFFKE, H. KLINGBEIL, M. KUMM, and M. GLESNER. Implementation of Realtime and Highspeed Phase Detector on FPGA. In K. BERTELS, J. M. P. CARDOSO, and S. VASSILIADIS, eds., *Reconfigurable Computing: Architectures and Applications, ARC '06*, volume 3985 of *Lecture Notes in Computer Science*, pages 1–11. Springer, Feb. 2006.

[136] H.-P. KEIL, M. MOMENI, A. GUNTORO, A. ORTIZ, and M. GLESNER. A Novel Leakage-Estimation Method for Input-Vector Control. In *Proc. of the Intl. Asia-Pacific Conference on Circuits and Systems, APCCAS '08*, Dec. 2008.

[137] M. MOMENI, A. GUNTORO, H.-P. KEIL, and M. GLESNER. Impact of Circuit Nonidealities on the Implementation of Switched-Capacitor Resonators. In *Proc. of the Intl. Asia-Pacific Conference on Circuits and Systems, APCCAS '08*, Dec. 2008.

[138] M. MOMENI, A. GUNTORO, H.-P. KEIL, and M. GLESNER. Influence of Circuit Nonidealities on Switched-Capacitor Resonators. In *Proc. of the Intl.Conference on Signals and Electronic Systems, ICSES '08*, pages 93–96, Sep. 2008.

[139] T. MURGAN, A. GUNTORO, H. HINKELMANN, P. B. BACINSCHI, and M. GLESNER. Low-Complexity Adaptive Encoding Schemes Based on Partial Bus-Invert for Power Reduction in Buses Exhibiting Capacitive Coupling. In G. SASSATELLI, M. GLESNER, C. BOBDA, and P. BENOIT, eds., *Workshop on Reconfigurable Communication-centric SoCs, ReCoSoC '07*, pages 7–14. Univ. Montpellier II, June 2007.

[140] T. MURGAN, A. M. OBEID, A. GUNTORO, P. ZIPF, M. GLESNER, and U. HEINKEL. Design and Implementation of a Multi-Core Architecture for Overhead Processing in Optical Transport Networks. In G. SASSATELLI, M. GLESNER, L. TORRES, L. S. INDRUSIAK, and T. HOLLSTEIN, eds., *Workshop on Reconfigurable Communication-centric SoCs, ReCoSoC '05*, pages 151–156. Univ. Montpellier II, June 2005.

[141] THIANG, A. GUNTORO, and R. LIM. Type of Vehicle Recognition Using Template Matching Method. In *Proc. of the International Conf. on Electrical, Electronics, Communication and Information, CECI '01*, May 2001.

[142] THIANG, R. LIM, and A. GUNTORO. Car recognition using Gabor filter feature extraction. In *Proc. of the Intl. Asia-Pacific Conference on Circuits and Systems, APCCAS '02*, volume 2, pages 451–455. IEEE, Dec. 2002.

Supervised Theses

[143] H. CHOKR. Untersuchung, Vergleich und FPGA Implementierung verchiedener DCT- und DWT-Algorithmen für Bildverarbeitung. Masterarbeit, TU Darmstadt, Aug. 2007.

[144] J. DENG. Filter Design and Implementation for Realtime Applications. Studienarbeit, TU Darmstadt, May 2005.

[145] O. GAMERO. Reassembling the smt374 firmware to ease the add-ons module attachment. Studienarbeit, TU Darmstadt, Oct. 2007.

[146] L. JU. Design methodology of partial reconfiguration in fpga-based system. Master thesis, University of Applied Sciences Darmstadt, May 2006.

[147] E. LOUAI. Analysis of the System-on-Chip Bus Architecture for Partial Reconfiguration Support in FPGA. Bachelorarbeit, TU Darmstadt, Dec. 2006.

[148] O.-C. LUCA-SAVIN. Implementation of Trigonometry Functions inside the FPGA for the Realtime Applications. Diplomarbeit, The Technical University of Iasi, July 2005.

[149] E. NGUH. Hardware implementation of cdf wavelet filters using lifting algorithm. Master thesis, TU Darmstadt, Aug. 2008.

[150] M. RAMADAN. Entwicklung eines eingebetteten Linux Systems für die dynamisch partielle Selbstrekonfiguration. Diplomarbeit, TU Darmstadt, Feb. 2007.

[151] Y. SUO. Analyse, Entwurf und Implementierung der arithmetische Division unter VHDL. Studienarbeit, TU Darmstadt, Oct. 2005.

[152] A. THEISEN. Implementierung, Optimierung und Systemintegration eines FIR Filters in VHDL. Bachelorarbeit, TU Darmstadt, June 2007.

Curriculum Vitæ

Andre T. GUNTORO

Personal Data:

Date of birth:	July 13th, 1979
Place of birth:	Indramayu, Indonesia

Academic Formation:

1984 - 1993	Primary (*SD*) and lower secondary school (*SMP*) at the BPK Penabur in Indramayu, Indonesia
1993 - 1996	Upper secondary school (*SMA*) at the SMA Negeri 1 in Indramayu, Indonesia
1996 - 2000	Student at the Faculty of Electronics, Petra Christian University, Surabaya, Indonesia Degree: Bachelor of Science
2001 - 2003	Student at the Darmstadt University of Applied Science with major in Telecommunication, Germany Degree: Master of Science
2002 - 2004	Student at the Technische Universität Darmstadt with major in Information and Communication Engineering, Germany Degree: Master of Science
2004 - 2009	Ph.D. student, teaching and research assistant at the Institute of Microelectronic Systems, Technische Universität Darmstadt, Germany